UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

SYNESTESIA: Nuevas formas de sentir la música

Juan José Daza Linares

Tutor: José Manuel Merino Álvarez

JUNIO 2021

SYNESTESIA: Nuevas formas de sentir la música

AUTOR: Juan José Daza Linares

TUTOR: José Manuel Merino Álvarez

Ingeniería Informática

Dpto. Ingeniería Informática

Escuela Politécnica Superior
Universidad Autónoma de Madrid

Junio de 2021

Resumen

Nadie cuestionaría que la música es una parte importante de nuestras vidas; presente en el ocio, en el trabajo y pilar fundamental de la publicidad. Muchos no podrían vivir sin ella, sin embargo, algunos no tienen elección. El objetivo de este Trabajo de Fin de Grado es desarrollar un sistema ***central*** y ***escalable,*** que permita sentir la música de nuevas formas y brindar la oportunidad de apreciarla a aquellas personas con dificultades auditivas.

Para facilitar el acceso al sistema, éste ha sido desarrollado utilizando componentes de bajo coste y accesibles. El lenguaje escogido para el desarrollo del software es C. Dada su fácil comprensión, la eficiencia y la gestión de recursos que ofrece, se convierte en la opción ideal para sistemas que no disponen de una gran cantidad de recursos y donde el rendimiento es crítico.

El sistema desarrollado está compuesto por dos tipos de dispositivos, el dispositivo *emisor* y los dispositivos *cliente*.

- El dispositivo emisor es el encargado de recoger las muestras sonoras y, mediante la Transformada de Fourier, reconocer la frecuencia predominante entre el ruido. Dicha frecuencia es retransmitida a los dispositivos cliente conectados, haciendo uso de la dirección IP de broadcast. Para que el dispositivo emisor pueda retransmitir la frecuencia leída, es necesario que actúe como un punto de acceso al que conectarse. Además, el dispositivo emisor es capaz de gestionar simultáneamente un módulo sensorial. El módulo sensorial y los credenciales del punto de acceso pueden ser configurados desde la aplicación web, gestionada por el propio dispositivo, donde, también, es posible persistir la configuración para cargarla en inicios posteriores.
- Los dispositivos cliente se conectan a un dispositivo emisor y reciben la frecuencia leída por éste automáticamente, sin la necesidad de solicitarla. Al igual que el dispositivo emisor, este tipo de dispositivo actúa como punto de acceso e incorpora una aplicación web, donde, además de configurar los credenciales del punto de acceso y la configuración del módulo, es posible introducir los credenciales que permiten establecer una conexión con el punto de acceso del emisor.

Para mostrar las capacidades de este sistema, se ha desarrollado un prototipo del dispositivo emisor, conectado a un módulo sensorial, y un dispositivo cliente, conectado a otro módulo de iguales prestaciones, en concreto, un juego de luces *LED*.

Palabras clave

Sinestesia, Música, Sentir, Sentidos, Audio, Arduino, FFT, Frecuencias, Notas, Broadcast, UDP

Abstract

No one would ever doubt that music is a substantial part of our life; it is used to recreate ambiances, to relax and it is even used to influence people's perceptions of a product. While many cannot live without it, some are unable to experience it at all. This bachelor thesis introduces an upgradable system that introduces new ways of experiencing music, enabling those who suffer disabilities that prevent them from hearing music, to feel it.

This system is made with accessibility in mind, by using low cost and common components. It is developed in *C*, it is easy to understand, resource-friendly and efficient, which makes it ideal for low-specs systems.

There are two kind of devices, an emitter and a receiver:

- The emitter device takes the sound samples to analyze, applies the Fourier Transform to it and retrieves the main frequency of the sound. This frequency is broadcasted to all connected receiver devices. In order to accomplish this broadcasting, the emitter device must act as an access point. Additionally, this type of device allows the user to add a sensitive module, which can be configured through the web app, as well as the access point settings. Both configurations can be saved through the web app, so they can be loaded at boot.
- The receiver devices connect to an emitter device and receive the broadcasted frequency, without making any request. Just like the emitter device, the receiver devices act as an access point, which means that users can enter the web app and configure the main and module parameters.

To demonstrate this system's capacities one prototype of each kind have been developed, both connected to a sensitive module

Keywords

Synesthesia, Music, Feel, Senses, Audio, Arduino, FFT, Frequencies, Musical Notes, Broadcast, UDP

ÍNDICE DE CONTENIDOS

INDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE ILUSTRACIONES

ÍNDICE DE CÓDIGOS

ÍNDICE DE ECUACIONES

Agradecimientos

Es necesario reconocer y agradecer a José Manuel Merino su interés y su ayuda para encauzar esta idea, desde su concepto primitivo e inconexo, hasta el prototipo que es hoy en día. Este proyecto tampoco hubiera sido posible sin la colaboración de Alejandro Braña, quien ha señalado los riesgos durante el desarrollo, las mejores opciones para la selección de componentes, y ayudado a realizar las primeras conexiones del prototipo desarrollado. Muchas gracias por todo tu tiempo invertido y por compartir tu experiencia, he aprendido mucho.

Gracias, también, a Abhilash Patel, un apasionado de la tecnología que ha facilitado la librería para aplicar la Transformada de Fourier rápida y eficazmente.

1 Introducción

¿De qué color es la letra "A"? ¿Roja [1]? ¿Por qué? ¿Cómo es posible que podamos asociar una letra a un color? Esto se conoce como sinestesia, la capacidad que tienen algunas personas para experimentar distintas sensaciones ante un mismo estímulo. Pero, ¿por qué no explotarlo? ¿Por qué no entrenar nuestros sentidos? O, incluso, ¿por qué no estimular un sentido con los estímulos propios de otro?

1.1 *Motivación*

Como se mencionaba en el resumen de este documento, la música es una parte importante de nuestra vida; cobra vital importancia en la comunicación del ser humano, ya que, como otras artes, es un método de expresión global, entendido mundialmente, con el que somos capaces de mostrar sentimientos que, de otra forma, son indescriptibles. Sin embargo, se trata de un "idioma" que no todos pueden disfrutar.

Como músico, creo en la importancia de la expresión mediante la música y, considero que ésta puede percibirse por más sentidos además del auditivo. Un claro ejemplo de esto se encuentra en los (ya casi olvidados) conciertos, donde, cada actuación está acompañada por un espectáculo de luces que sumerge al espectador en la dimensión de la música.

Si logramos que una letra tenga un color, ¿por qué no una nota musical? La motivación de este Trabajo nace de la idea de juntar sentidos, potenciarlos y brindar la oportunidad de relacionar la música con mucho más que el sentido auditivo. Además, este proyecto abarca otros muchos usos, como disponer de un espectáculo de luces en espacios reducidos y sin medios o aprender a tocar un instrumento sin saber leer partituras, apoyándonos en la vista, por ejemplo.

[1] N. B. Root, R. Rouw, M. Asano, C.-Y. Kim, H. Melero, K. Yokosawa y V. S. Ramachandran, «Why is the synesthete's "A" red? Using a five-language dataset to disentangle the effects of shape, sound, semantics, and ordinality on inducer–concurrent relationships in grapheme-color synesthesia,» *ScienceDirect,* vol. 99, pp. 375-389, 2018.

1.2 *Objetivos*

El objetivo final de este Trabajo de Fin de Grado es desarrollar un sistema que realice el reconocimiento de las frecuencias producidas por un instrumento, su gestión y la estimulación de algún sentido de acuerdo a la frecuencia reconocida. El sistema deberá ser configurable en su totalidad, sin la necesidad de un ordenador. Además, este sistema debe ser **económico**, **fácil de usar**, de **bajo consumo** y **escalable**. Dada la naturaleza de este proyecto, el sistema define dos tipos de dispositivos: emisor y cliente. Se realizará un prototipo de cada tipo de dispositivo, usable en su totalidad.

1.3 *Organización de la memoria*

En esta memoria se documenta el proceso de desarrollo de los dispositivos prototipo, dividido en 7 capítulos:

1. **Introducción**: Este capítulo describe la motivación del Trabajo, sus objetivos y la disposición del documento.
2. **Estado del arte**: En este capítulo se introducen las tecnologías que permiten el reconocimiento de frecuencias, así como, algunos proyectos de interés y las tecnologías empleadas en el proyecto.
3. **Planning**: En este capítulo se detallan la metodología usada y la organización (*planning*) seguidas durante el desarrollo de este Trabajo de Fin de Grado.
4. **Diseño**: Este capítulo contiene la descripción de los componentes hardware y las conexiones entre ellos, además, incluye el detalle de los componentes software desarrollados, junto a las decisiones de diseño realizadas.
5. **Implementación**: En este capítulo se detallan la integración de todos los componentes, el funcionamiento de cada tipo de dispositivo y los programas a ejecutar para el uso del sistema.
6. **Pruebas y Resultados**: En este capítulo se recoge la batería de pruebas unitarias realizadas a los componentes, además, incluye la descripción de las pruebas realizadas al sistema completo, junto a la evaluación de los resultados obtenidos.
7. **Conclusiones y trabajo futuro**: En este capítulo se hace una reflexión sobre el sistema desarrollado y, en extensión, se indica el trabajo a realizar en futuros desarrollos.

Además, se han incluido los siguientes anexos:

A. Manual de Instalación.
B. Tareas por *sprint.*
C. Aplicación web.
D. Conversión de frecuencias a nota musical.
E. Capturas de paquetes con Wireshark.

2 Estado del arte

En este capítulo se realiza una breve recopilación de las tecnologías que permiten realizar el reconocimiento y procesado digital de las frecuencias musicales. Además, se introducirán los microcontroladores que dan soporte a este proyecto, junto a algunos proyectos de interés, y las limitaciones que presentan. Por último, al final de este capítulo, se presenta el conjunto de librerías que hacen posible el desarrollo del software de este proyecto.

2.1 *Análisis de sonido*

2.1.1 Micrófonos

El primer paso para reconocer una frecuencia musical es precisar del dispositivo con el que recogerlas. Para ello, se hace uso de micrófonos, unos dispositivos capaces de recoger las vibraciones producidas por las ondas de sonido y convertirlas en una variación de corriente o tensión, conservando sus propiedades originales.

Micrófonos de condensador

Los micrófonos de condensador están principalmente compuestos por un condensador de dos placas metálicas, una fija y otra móvil, pegada al diafragma. Cuando el sonido entra en contacto con el diafragma hace que éste vibre y, por tanto, se modifica la distancia entre las dos placas. Cuando la distancia entre las placas del condensador es alterada también se altera su capacidad, a la misma frecuencia con la que el sonido golpea el diafragma, produciendo un cambio en la tensión del circuito al que esté conectado. Es importante mencionar que, para que este micrófono funcione debe estar alimentado, de lo contrario, no será posible obtener un valor a la salida, puesto que no circula corriente por él.

Estos micrófonos proporcionan una gran sensibilidad y trabajan en un rango de frecuencias muy amplio, entre los 20 y 20.000 Hz, además, son económicos y comunes.

2.1.2 Amplificadores

La variación de corriente (o tensión) producida por un micrófono es del orden de los miliamperios (o milivoltios), por lo que es común amplificarla. Para ello, se realiza uso de los circuitos amplificadores, unos circuitos capaces de incrementar la señal de salida con la ayuda de una fuente de alimentación externa.

2.1.3 Transformada de Fourier [1]

Idealmente, las muestras tomadas por el micrófono consistirían en un sonido limpio, formado por una única frecuencia, pero, desafortunadamente, la muestra capturada estará formada por la suma de varias frecuencias, por lo que será necesario separarlas mediante la Transformada de Fourier.

La Transformada de Fourier se trata de una transformación matemática que permite, no sólo saber qué frecuencias se unen para formar un sonido, sino en qué cantidad, es decir, con qué amplitud. Por ejemplo, supongamos que la señal leída por el micrófono es la representada en la figura 2.3. Tal como se ha mencionado anteriormente, esta señal no será la representación de un sonido limpio, sino la suma de dos o más señales producidas por varias fuentes. La Transformada de Fourier permite obtener las señales representadas en las figuras 2.4 y 2.5, las señales originales cuya suma produjo la señal captada por el micrófono.

La aplicación de la Transformada de Fourier requiere de diversas operaciones complejas que necesitan una alta capacidad de cómputo, como la multiplicación de varios valores de coma flotante (cálculos de senos y cosenos).

2.1.4 Implementaciones de la Transformada de Fourier

A continuación se mencionarán dos de las implementaciones de la Transformada de Fourier diseñadas para reducir su complejidad y el uso de memoria.

FFT (Fast Fourier Transform) [2]

Este algoritmo, desarrollado por James Cooley y John Tukey en 1965, proporciona la aplicación de la Transformada de Fourier mediante recursión, obteniendo una complejidad ***O(N log N).*** Esta implementación sigue la filosofía "*divide y vencerás"*, la cual supone descomponer una tarea computacionalmente costosa en otras de menor complejidad.

Figura 2.1: Simulación de una señal compuesta de dos frecuencias, *f* y *f'*

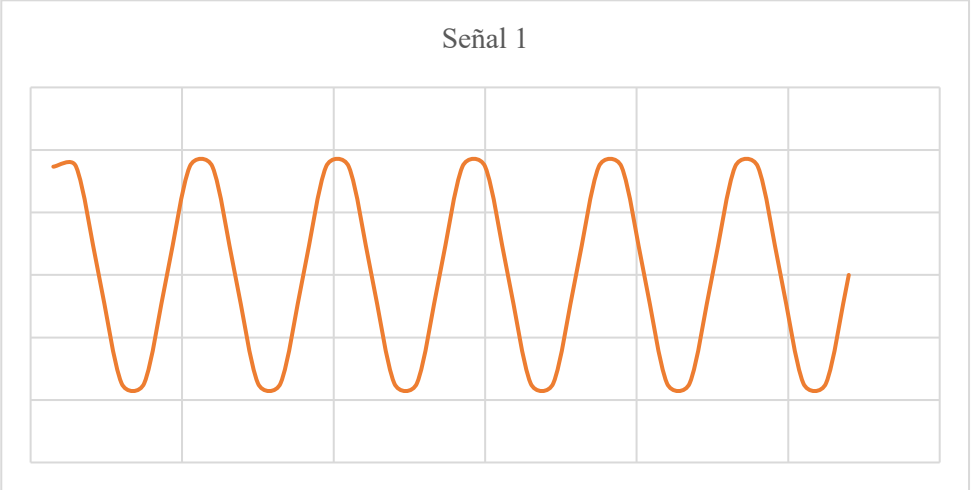

Figura 2.2: Simulación de una señal de frecuencia, *f*

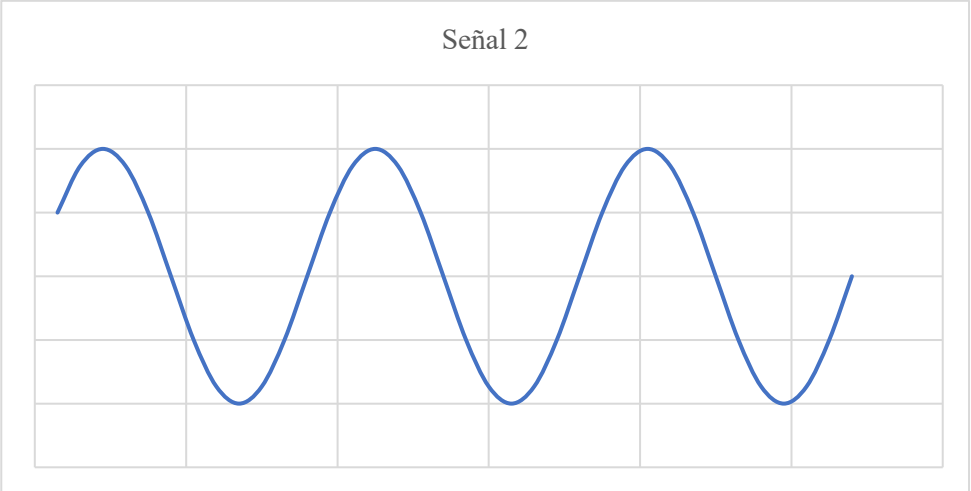

Figura 2.3: Simulación de una señal de frecuencia, *f'*

ApproxFFT [3]

Con el auge de los controladores Arduino, se han desarrollado numerosas implementaciones que, incluso con las limitaciones de estos sistemas, aplican la Transformada de Fourier rápidamente. Este es el caso del algoritmo *ApproxFFT*, desarrollado por Abhilash Patel en diciembre de 2020.

Este algoritmo aplica la Transformada de Fourier hasta tres veces más rápido que el algoritmo *FFT*, como puede apreciarse en la tabla 2.1**,** donde se muestran los tiempos de ejecución de cada algoritmo en función del número de muestras tomadas, sobre un controlador *Arduino Nano*.

Número de muestras	FFT (ms)	ApproxFFT (ms)
16	7	2
32	17	4
64	37	9
128	80	21
256	174	55
512	395	129
1024	No es posible	271

Tabla 2.1: Comparación de tiempos entre FFT y ApproxFFT

A diferencia de otras implementaciones, este algoritmo hace uso de funciones seno/coseno *pre-calculadas*, lo que evita calcular el seno/coseno para cada una de las muestras. Como es de esperar, este hecho reduce, ligeramente, la precisión del análisis.

2.1.5 Teorema de Nyquist-Shannon [4]

El teorema de Nyquist-Shannon indica cuál debe ser la tasa mínima de muestreo para reconocer una frecuencia. En concreto, este teorema establece que la tasa de muestreo debe ser el doble de la frecuencia más alta a reconocer; por ejemplo, si se desea reconocer una frecuencia máxima de 1KHz, la tasa de muestreo debe ser, como mínimo, de 2KHz.

Para captar correctamente la frecuencia de una señal es necesario **conocer a qué velocidad oscila** entre sus valores positivos y negativos. Cuando la tasa de muestreo es superior al doble de la frecuencia analizada, es posible representar su oscilación con varias muestras por periodo, como puede apreciarse en la figura 2.4. Sin embargo, a medida que la frecuencia aumenta, el número de muestras por periodo disminuye, como se muestra en la figura 2.5. Cuando la frecuencia de la señal es justo la mitad que la tasa de muestreo, el número de muestras por periodo se limita a dos, una en el máximo y otra en el mínimo de la señal, como ocurre en la figura 2.6. Para frecuencias mayores, hay menos de dos muestras por periodo y, por tanto, no es posible representarla.

Figura 2.4: Tasa de muestreo 8 veces superior a la frecuencia analizada.

Figura 2.5: Tasa de muestreo 4 veces superior a la frecuencia analizada.

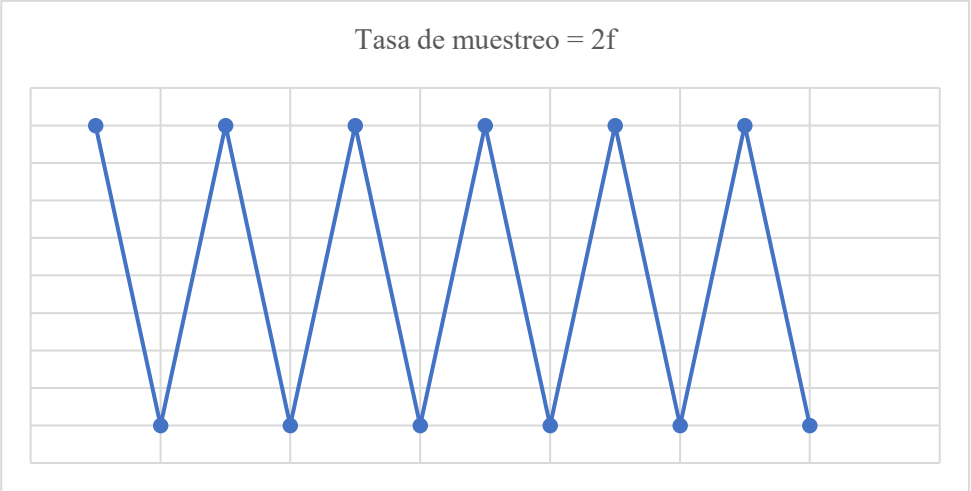

Figura 2.6: Tasa de muestreo 2 veces superior a la frecuencia analizada.

2.2 *Microcontroladores basados en Arduino [5]*

Los microcontroladores basados en *Arduino* se tratan de circuitos integrados, embebidos en placas entrenadoras que siguen un estándar de hardware y software libres, con el nombre de *Arduino*. En los últimos años su uso ha aumentado considerablemente, propiciando el desarrollo de nuevas placas y funcionalidades. A continuación, se mencionan algunas propiedades fundamentales de los microcontroladores utilizados durante el desarrollo de ese Trabajo de Fin de Grado.

2.2.1 ATmega328P [6]

El microcontrolador *ATmega328P* se trata de un microcontrolador de 8 bits, con una velocidad de *CPU* aproximadamente igual a 20MHz. Dispone de 23 líneas generales destinadas para entrada y salida (*GPIO*) y, requiere de una tensión de entre los 1.8 y 5.5 V para funcionar.

2.2.2 ESP8266 [7]

El microcontrolador *ESP8266* se trata de un microcontrolador de 32 *bits*, con una velocidad de *CPU* de 80 MHz. Dispone de 17 líneas generales destinadas para la entrada y salida (*GPIO*) y, requiere de una tensión de entre 2.5 y 3.6 V para funcionar. Además, este microcontrolador incorpora capacidades inalámbricas, para interactuar y crear redes de 2.4 GHz, que alcanzan velocidades de hasta 72 mbps. Por último, este microcontrolador implementa los protocolos *TCP/IP* y el estándar *802.11*.

2.3 *Reconocimiento de frecuencias en Arduino*

Dada la popularidad de los controladores *Arduino* es fácil encontrar proyectos de todo tipo, incluyendo aquellos diseñados para reconocer sonidos y analizar frecuencias. A continuación, se mencionan brevemente algunos proyectos de interés, dado el ámbito de este proyecto.

2.3.1 ESP32_FFT_VU [8]

ESP32_FFT_VU consiste en un visualizador de espectro de audio en tiempo real, desarrollado en el año 2020. Este proyecto recoge varias muestras de sonido y las distingue por rangos de frecuencias. A continuación, evalúa la amplitud de cada frecuencia reconocida y, en función del valor de amplitud total de cada rango, se enciende una cantidad variable de *LEDs*, mostrando visualmente la amplitud de cada grupo de frecuencias.

Además, el sistema es capaz de mostrar varias interfaces para visualizar las amplitudes, mediante la acción de un pulsador.

El proyecto tiene una función limitada a la evaluación de las frecuencias y la iluminación de luces *LEDs* y no ofrece opciones de configuración sin codificación, sin embargo, dada su sencillez y alta

legibilidad ha sido de utilidad para asentar las bases del tratamiento del señales acústicas en estos controladores.

2.3.2 ColorChord [9]

ColorChord se trata de un proyecto reciente, iniciado en el año 2010. Este proyecto se desarrolló para realizar la asignación de un color a una nota musical, mediante un módulo de luces *LED*.

Para realizar el reconocimiento de frecuencias el sistema recoge entre 512 y 1024 muestras por segundo y, posteriormente, aplica la Transformada de Fourier Discreta (*DFT* [10]), para la cual, es necesario definir una ventana (*window*) sobre la muestra; en este caso, los límites están definidos por el primer golpe de sonido y por su interrupción.

Tras el reconocimiento de las frecuencias que forman la muestra, se eliminan aquellas de menor amplitud, interpretadas como ruido y se transportan al espacio cromático de la ilustración 2.1. Esto, supone un gran inconveniente, ya que un Do_4 y un Do_6, serán interpretados como un Do, simplemente, a pesar de pertenecer a octavas diferentes.

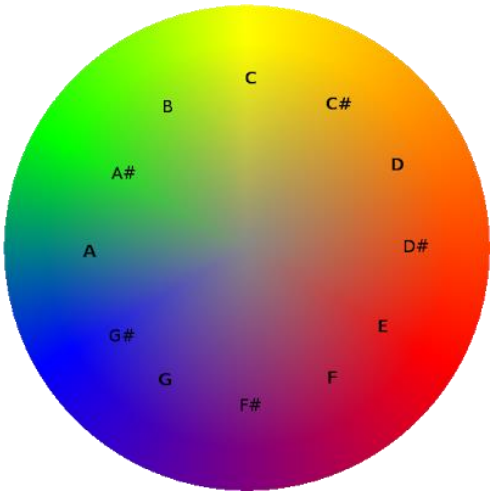

Ilustración 2.1: Espacio cromático de una octava**.** *Extraído de* **[9]**

La aplicación está desarrollada para sistemas de escritorio, en concreto, *Windows* y *Ubuntu*. Dado el alto consumo de recursos, los procesadores disponibles en el mercado por esas fechas no eran capaces de procesar las señales en tiempo real y, por tanto, el procesamiento de las señales requiere de una tarjeta gráfica dedicada.

Esta aplicación aporta una gran cantidad de funcionalidad, como limitar el número de *LEDs* que se encienden por cada nota y a qué intensidad, dependiendo de la fuerza con la que se ha producido el sonido, sin embargo, presenta varias limitaciones, como la necesidad de equipos potentes y dedicados, la escasez de configuración y la complejidad del sistema, que requiere de un nivel de conocimiento medio para, tan sólo, saber cómo ejecutarla. Además, como se ha comentado anteriormente, dado el

tipo de procesado que se realiza sobre las señales, es imposible realizar la diferenciación entre octavas, limitando las aplicaciones del sistema.

2.3.3 ColorChord 2 [11]

En el año 2016, *ColorChord* quedó obsoleto con la salida de *ColorChord 2*, una nueva aplicación diseñada bajo los mismos principios, aunque desarrollada desde cero para reducir considerablemente el uso de los recursos. Entre sus novedades se incluyen el análisis de las frecuencias en profundidad y nuevas formas de mostrar los sonidos, mediante figuras y colores varios, representados en la aplicación. Además, la novedad que resulta de mayor interés para este Trabajo de Fin de Grado, es la posibilidad de ejecutar *ColorChord 2* en dispositivos embebidos, como el microcontrolador *ESP8266*.

Como este proyecto se encuentra aún en desarrollo y no dispone de la funcionalidad completa, la documentación resulta confusa y no detalla las limitaciones o el alcance del proyecto completo. No obstante, el desarrollo actual del sistema en dispositivos embebidos permite controlar individualmente el color y la intensidad de cada *LED* contenido en un módulo *LED* del tipo *WS2812Bs* [2]. Además, la información de las frecuencias reconocidas se encuentra disponible en una aplicación web, en ejecución por el controlador.

La instalación del sistema en los dispositivos embebidos requiere de conocimientos avanzados, junto a un elaborado trabajo previo. Además, dada la especialización del proyecto, *ColorChord 2* comparte muchas de las limitaciones de su predecesor, como la nula configuración de colores, o la imposibilidad de procesar notas distribuidas en diferentes octavas.

2.4 *Tecnologías empleadas*

En este proyecto se ha hecho uso de varias librerías que compatibilizan y facilitan el desarrollo del sistema principal en los dispositivos empleados. A continuación, se mencionan algunas de las tecnologías utilizadas:

2.4.1 Lenguaje Arduino

El lenguaje *Arduino* se trata de un lenguaje de programación diseñado para operar fácilmente con las entradas y salidas de un controlador. Se trata una abstracción de *Wiring*, un lenguaje de programación orientado a microcontroladores.

[2] Este tipo de módulos LED son controlados por un microcontrolador integrado, al que se puede indicar el color y la intensidad de cada led que forma el módulo.

2.4.2 *Arduino ESP8266 Core* [12]

Arduino ESP8266 Core se trata de un proyecto que reúne la funcionalidad necesaria para adaptar y permitir el uso del lenguaje *Arduino* (y sus librerías) con los microcontroladores *ESP8266* embebidos. Además, el proyecto incluye otras muchas librerías que exprimen las capacidades de éste módulo *WiFi*.

El proyecto *Arduino ESP8266 Core* incluye varias librerías de utilidad, de entre las cuales, ha sido necesario el uso de las siguientes:

ESP8266WiFi [13]

Esta librería habilita el intercambio de información de forma inalámbrica en el módulo *ESP8266*. Dispone de dos modos, compatibles entre sí, con los cuales es posible recibir conexiones externas (mediante la creación de un punto de acceso) y realizarlas, actuando como un cliente conectado a una red *WiFi*.

WiFiUDP [14]

La librería *WiFiUDP* facilita la gestión de los *sockets UDP*, así como la creación y el envío/recepción de los paquetes *UDP*, haciendo transparente todo el proceso de configuración.

ESP8266WebServer [15]

Esta librería facilita el procesado de las peticiones *HTTP*, identificando el tipo de petición, administrando el contenido de las cabeceras y ejecutando los manejadores (*handler*) definidos para cada ruta durante la configuración.

DNSServer [16]

La librería *DNSServer* permite ejecutar un pequeño servidor de nombres en el módulo *ESP8266*, con el que es posible traducir nombres de dominios a las direcciones *IP* previamente asociadas, durante la configuración.

FS [17]

Esta librería facilita la gestión de los archivos contenidos en la memoria flash del controlador, proporcionando la lectura y escritura de ficheros como si se tratara de un sistema de archivos tradicional.

2.4.3 ArduinoThread [18]

La librería *ArduinoThread* simula la ejecución de tareas paralelas en hilos, algo no soportado en *Arduino*, mediante la técnica *Protothreading*.

2.4.4 Vue [19]

Vue se trata de un *framework* de Javascript orientado a la generación de *HTML* dinámico, para su uso en *SPA*.

2.4.5 Bootstrap [20]

Bootstrap se trata de un conjunto de estilos *CSS* que permite la adaptación automática de los componentes *HTML* a diferentes tamaños de pantalla. Además, proporciona colores, estilos y formas diseñados para mejorar la usabilidad.

3 Planning

Desarrollar el sistema completo conlleva muchos riesgos. Algunos pueden ser previstos y solucionados antes de comenzar el desarrollo, como la incompatibilidad entre componentes, librerías obsoletas o anticuadas e incluso el deterioro de algún componente principal . No obstante, otros tantos aparecen durante el desarrollo, pudiendo comprometer la entrega final y el Trabajo de Fin de Grado en su completitud. Por tanto, para reducir el impacto de los riesgos imprevistos, se ha optado por seguir una metodología ágil, con *sprints* de dos semanas, en los que, al terminar, se presenta una parte funcional del proyecto.

En concreto, la metodología ágil escogida ha sido basada en *Kanban*, y, como soporte para llevarla a cabo, se ha utilizado la versión gratuita del software *Jira*. Además, para gestionar las versiones del producto entregable y mantener separados los desarrollos, se ha utilizado la herramienta *git*, siguiendo el modelo *GitFlow,* en el que, cada rama tiene un código que lo relaciona con una tarea de *Jira*.

El desarrollo total del sistema se ha realizado en cinco *sprints,* dedicados, en su mayoría, al desarrollo de nueva funcionalidad, aunque al final de cada uno, se han llevado labores de mantenimiento, perfectivo y correctivo. Estos *sprints* quedan ilustrados con un *diagrama de Gantt* en la ilustración 3.1. Además, en el anexo B, se detalla la lista de tareas realizadas.

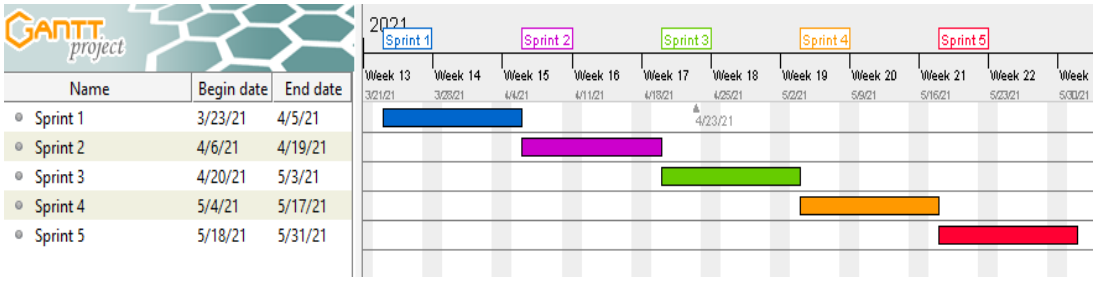

Ilustración 3.1: Distribución de los *sprints*.

3.1 *Objetivos por sprint*

3.1.1 Sprint 1

El sprint 1 tiene como objetivo realizar el desarrollo del software específico para controlar el color y la intensidad del juego de luces conectados al controlador. Este sprint se completó sin contratiempos.

3.1.2 Sprint 2

El *sprint* 2 tiene como objetivo realizar la implementación del sistema de gestión de archivos, con las opciones de lectura y escritura, además de la carga de ficheros desde un ordenador. Además, en este sprint se incluye la implementación de los componentes de configuración del dispositivo y del módulo sensorial. Este *sprint* no pudo completarse a tiempo y, el componente de configuración del módulo sensorial, tuvo que ser continuado en el *sprint* 3.

3.1.3 Sprint 3

El *sprint* 3 tiene como objetivo terminar el desarrollo no completado del sprint 2, además de desarrollar la aplicación web y los componentes que permitirán acceder a las configuraciones desde ésta. Por último, se debe realizar el software que persista las configuraciones en los archivos de configuración.

Este *sprint* no pudo completarse a tiempo y, el software que permite persistir las configuraciones, tuvo que ser continuado en el *sprint* 4.

3.1.4 Sprint 4

El *sprint* 4 tiene como objetivo terminar el desarrollo no completado del *sprint* 3 y desarrollar los componentes que permiten tomar las muestras de sonido y analizarlas para obtener la frecuencia predominante, así como el servicio de retransmisión y recepción de frecuencias.

Este *sprint* no pudo completarse a tiempo y, el servicio de retransmisión y recepción de frecuencias, tuvo que ser continuado en el *sprint* 5.

3.1.5 Sprint 5

El *sprint* 5 tiene como objetivo terminar el desarrollo no completado del *sprint* 4, desarrollar el software principal que agrupe los componentes desarrollados durante los *sprint* anteriores y, realizar el montaje definitivo de los prototipos. Este *sprint* se completó sin contratiempos.

4 Diseño

En este capítulo se recoge toda la información relacionada con el diseño y desarrollo del sistema, en concreto, los requisitos, la elección de componentes hardware y sus conexiones, y, por último el desarrollo de los componentes software.

4.1 *Requisitos*

En esta sección se detallan los requisitos funcionales y no funcionales del proyecto.

4.1.1 Requisitos funcionales

General

- **RF-1**.: El sistema debe permitir la definición y convivencia de dos tipos de dispositivos, emisor y cliente.
- **RF-2**.: El sistema debe permitir el uso de un módulo sensorial.
- **RF-3**.: El sistema debe funcionar correctamente aunque no se use un módulo sensorial.
- **RF-4**.: El sistema debe persistir la configuración en memoria.
- **RF-5**.: El sistema debe cargar la configuración almacenada en memoria durante el inicio.
- **RF-6**.: El sistema debe permitir la modificación de la configuración mediante una aplicación web.
- **RF-7**.: El sistema incorporará un módulo sensorial, formado por un conjunto de luces *LED*.

Dispositivo emisor

- **RF-8**.: El dispositivo emisor debe reconocer las frecuencias producidas por el instrumento más cercano.
- **RF-9**.: El dispositivo emisor debe realizar la retransmisión de la frecuencia reconocida, en tiempo real, a todos los dispositivos conectados.

Dispositivo cliente

- **RF-10**.: El dispositivo cliente debe recibir, en tiempo real, la frecuencia reconocida por el dispositivo emisor.
- **RF-11**.: El dispositivo cliente debe poder conectarse a un dispositivo emisor.

4.1.2 Requisitos no funcionales

- **RNF-1.**: El sistema se realizará con componentes hardware de bajo consumo y fáciles de conseguir.
- **RNF-2.**: El sistema será escalable.
- **RNF-3.**: La aplicación web será intuitiva y accesible.
- **RNF-4.**: El sistema debe facilitar el desarrollo de módulos sensoriales.

4.2 *Elección de los componentes*

Como se ha comentado con anterioridad, este sistema ha sido desarrollado priorizando la accesibilidad, un bajo coste y la escalabilidad, por lo que, antes de proponer una solución software, ha sido necesaria la correcta selección de los componentes, la primera toma de contacto con el usuario final. Escoger adecuadamente los componentes facilitará que el sistema esté al alcance de todos los usuarios interesados, tanto aficionados como desarrolladores.

4.2.1 Controlador *NodeMCU ESP8266*

Este controlador está basado en *Arduino* e incluye un módulo con un microcontrolador *ESP8266* embebido. Este módulo hace del controlador *NodeMCU* la opción ideal, ya que permite gestionar conexiones inalámbricas entrantes y salientes al mismo tiempo, actuando como cliente y servidor, **simultáneamente**. Además de la facilidad que ofrece para gestionar otros aspectos inalámbricos, este controlador incorpora un sistema de ficheros en su memoria flash (de tipo *SPIFFS*), con escritura y lectura en tiempo de ejecución.

4.2.2 Controlador *Arduino Nano*

Dadas las limitaciones de recursos y *CPU* de los controladores *NodeMCU*, se ha optado por incorporar este conocido controlador, a modo de auxiliar, para distribuir la carga de trabajo que debe soportar un dispositivo emisor. De esta forma, es posible agilizar el sistema final. Este controlador aporta un microcontrolador *ATMega328p*, suficiente para realizar la toma de muestras de audio y su posterior procesado.

4.2.3 Micrófono *Adafruit MAX9814*

Esta interfaz de audio está formada por un micrófono de condensador y todos los componentes necesarios para amplificar la señal de tensión generada, sin la necesidad de añadir ningún filtro o circuito externo, lo que facilita y estandariza la comunicación de la interfaz con el controlador.

4.2.4 Módulo de luces *LED*

Para el desarrollo del módulo sensorial que se incorpora en este prototipo, se ha escogido un módulo de luces *LED RGB*.

4.2.5 Otros componentes

Para el correcto funcionamiento del módulo y poder controlar el color y la intensidad de cada color, ha sido necesario incluir tres transistores *2N22A*.

4.3 *Conexión entre los componentes*

La conexión de los componentes ha sido cuidadosamente diseñada para ocupar la menor cantidad de espacio posible, sin afectar a la modularidad de los componentes más frágiles y sujetos a cambios, como el micrófono y el módulo conectado.

Antes de realizar el diseño definitivo de las conexiones y, para verificar el correcto funcionamiento y comunicación de los componentes, el sistema ha sido desarrollado sobre una placa entrenadora. Tras haber definido y comprobado que todos los componentes respondieran según lo esperado, el sistema ha sido reorganizado en una placa *PCB* prototipo.

En las siguientes secciones se detallan los diagramas de conexiones del dispositivo emisor y del dispositivo cliente.

4.3.1 Esquema del dispositivo emisor

El dispositivo emisor requiere de un controlador *NodeMCU* y un controlador *Arduino Nano*, la interfaz de audio y, aunque es opcional para su funcionamiento, un módulo sensorial, en este prototipo, un módulo de luces *LED*.

4.3.1.1 Conexión entre los controladores

Es necesario recordar que el controlador *Arduino Nano* se utiliza a modo de apoyo, para distribuir la carga de trabajo del dispositivo emisor, por tanto, ambos dispositivos deben estar conectados entre sí y compartir información. Además, dada la naturaleza del controlador *Arduino Nano*, esta conexión debe ser alámbrica.

El protocolo de comunicación que mejor se adapta a las necesidades y características del dispositivo emisor es el protocolo I^2C [21] (*Inter-Integrated Circuit*), capaz de transmitir información rápidamente y sin pérdidas. Para implementar el protocolo I^2C únicamente es necesario conocer las líneas de conexión de datos (*SDA*) y sincronización (*SCL*) de ambos controladores y conectarlas entre sí, además de poner en común la conexión a tierra. Mientras que, en el controlador *NodeMCU* es posible

definir qué pines utilizar para cada línea, el *Arduino Nano* tiene un pin reservado para cada línea, en concreto, el pin *A4*, para la línea de datos (*SDA*) y, el pin *A5*, para la línea de sincronización (*SCL*).

Estos pines han sido conectados a los pines *D1* y *D2*, respectivamente, del controlador *NodeMCU*, aunque podrían haberse escogido otros pines diferentes; como se ha mencionado anteriormente, en este controlador, el pin reservado para cada línea se establece desde el software.

4.3.1.2 Conexión de la interfaz de audio

Como la interfaz de audio incorpora todos los elementos necesarios para amplificar, regular y controlar la tensión generada por los diferentes sonidos, no se debe de realizar ningún circuito regulador o amplificador y, por tanto, es posible conectarlo directamente al controlador *Arduino Nano*.

La interfaz de audio escogida se trata de un micrófono de condensador *Electret*, y requiere de una señal de 5V para funcionar. Además, ha sido configurada para funcionar con una ganancia de 40dB, mediante la conexión de la entrada *Gain* a la misma señal de tensión utilizada para alimentar la interfaz. Por último, la salida analógica de la interfaz ha sido conectada a la entrada analógica, *A1*, del controlador *Arduino Nano*.

4.3.1.3 Conexión del módulo de luces LED

El módulo de luces *LED* está formado por una única entrada y tres salidas, tal como se muestra, de forma simplificada, en la ilustración 4.1. Para iluminar un color del módulo de luces, es necesario conectar la entrada *Vin* al borne positivo de la fuente de corriente y, la salida del color deseado, al borne negativo o tierra. Si se deseara iluminar los tres colores al mismo tiempo, bastaría con conectar las tres salidas, simultáneamente, a tierra.

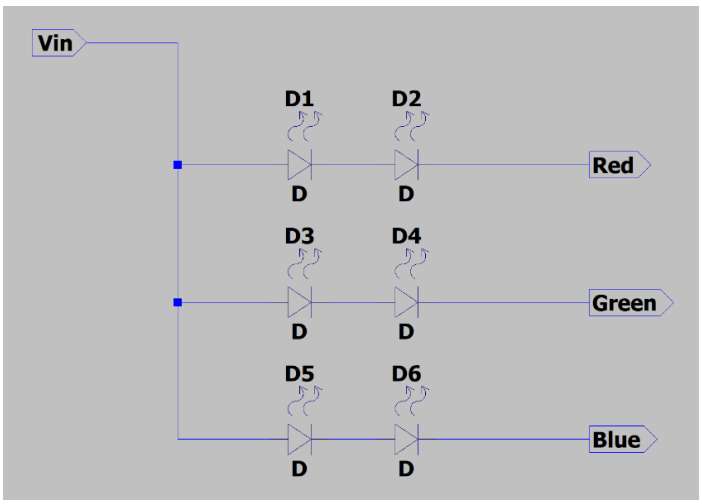

Ilustración 4.1: Diagrama de conexión del módulo de luces *LED*

No obstante, el módulo de luces, por sí mismo, no permite controlar el encendido o apagado de cada color y, mucho menos, variar la intensidad con la que se ilumina cada secuencia de *LEDs*. Por ello, es necesario añadir un pequeño circuito externo que, variando la tensión, sea capaz de regular la corriente que circula por cada secuencia de leds; un trabajo ideal para los transistores.

Un transistor es capaz de alterar la corriente que circula entre el emisor y el colector variando la diferencia de tensión entre su base y el emisor. Por tanto, para ajustar la intensidad con la que se ilumina una secuencia de leds, es necesario añadir un transistor a la salida de cada color, de la siguiente forma:

- El emisor del transistor ha de ser conectado a la salida del color
- El colector del transistor ha de ser conectado a tierra.
- La base del transistor debe ser conectada a una salida analógica del controlador

Las salidas analógicas a las que se conecten las bases de los transistores no es relevante, ya que pueden ser configuradas mediante la aplicación web.

4.3.1.4 Diagrama de conexiones del dispositivo emisor

El diagrama de conexiones final del dispositivo emisor queda reflejado en la ilustración 4.2.

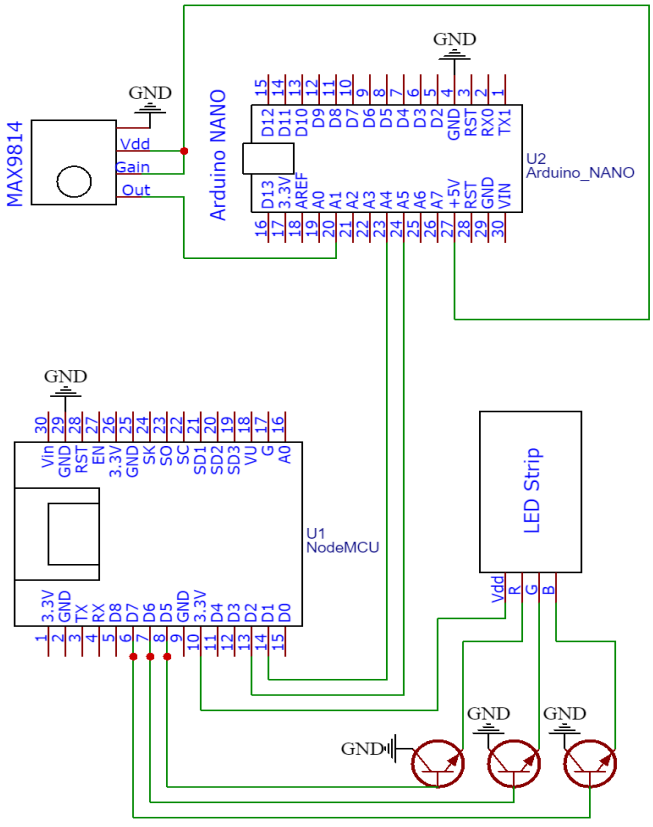

Ilustración 4.2: Diagrama de conexiones del dispositivo emisor.

4.3.2 Esquema del dispositivo cliente

El dispositivo cliente requiere de un controlador *NodeMCU* y un módulo sensorial, en este prototipo, un módulo de luces *LED*.

4.3.2.1 Conexión del módulo de luces LED

Como la conexión del módulo de luces *LED* del dispositivo cliente es idéntica a la del dispositivo emisor, el lector se puede referir a la ***sección 4.3.1.3***, para visualizar la conexión en detalle.

4.4 *Elección del lenguaje*

El primer paso para desarrollar los componentes que forman el sistema es escoger el lenguaje de programación en el que desarrollarlos; dependiendo de cuál se utilice, variará su diseño y su integración en el sistema. Como los controladores escogidos están basados en *Arduino* y, estos deben ser programados desde *Arduino IDE*, las opciones disponibles quedan reducidas a *C* y *C++*. Pese a que ambos lenguajes utilizan los recursos eficientemente y, tienen un rendimiento similar, se ha optado por escoger el lenguaje *C*, ya que es más fácil de comprender, al no estar orientado a objetos. No obstante, alguno de los componentes software utilizan librerías externas, desarrolladas en *C++*.

4.5 *Desarrollo de los componentes software*

El sistema se ha definido priorizando su escalabilidad y mantenibilidad. Para ello, se ha desglosado en componentes independientes que abstraen la complejidad total del sistema y facilitan su sustitución y reutilización. Estos componentes independientes pueden ser agrupados en dos grupos, *global* y *module*:

- Grupo *global*: A este grupo pertenecen los componentes de los que depende el sistema para funcionar correctamente.
- Grupo *module*: A este grupo pertenecen los componentes de los que depende el módulo sensorial para funcionar correctamente

Esta diferenciación puede realizarse ya que, tanto el dispositivo emisor como el dispositivo cliente, deben funcionar correctamente aunque no incorporen un módulo sensorial.

4.5.1 Componentes globales

4.5.1.1 file_manager

El componente *file_manager* está diseñado para gestionar la lectura y escritura de ficheros en la memoria flash del controlador, optimizando los recursos y recogiendo los posibles errores que pueden causar este tipo de operaciones. Este componente se trata de una abstracción de la funcionalidad proporcionada por la clase *SPIFFS* [3].

Además, se ha incorporado un método que, dada una secuencia de caracteres, obtiene el conjunto de pares clave-valor definidos, que facilitará la interpretación de las configuraciones.

4.5.1.2 audio

El componente *audio* está diseñado para recoger las señales analógicas producidas por el micrófono, y, posteriormente, aplicar la Transformada de Fourier, con el objetivo de obtener la frecuencia de la onda con mayor amplitud. La primera tarea de este componente, por tanto, consiste en la recogida de las señales analógicas, que debe tener en cuenta los siguientes factores:

- **Tasa de muestreo**: La tasa a la que se recogerá una muestra. Según el teorema de Nyquist-Shannon, la frecuencia máxima que puede ser detectada corresponde a la mitad de la tasa de muestreo.
- **Número de muestras por frecuencia**: El número de muestras que se deben tomar para caracterizar una frecuencia. Este número es directamente proporcional al tiempo de procesado y a la precisión de la caracterización.
- **Valor del ruido**: Corresponde al valor analógico producido por el micrófono cuando no detecta ningún sonido. Este valor puede ser restado a cada muestra tomada para descartar sonidos que no superen una amplitud.

Para el desarrollo de este proyecto, estos valores han sido seleccionados tomando una flauta travesera como referencia [4], sin embargo, para aumentar la compatibilidad con otros instrumentos, sin requerir modificaciones, se ha aumentado el margen de cada valor, resultando en los siguientes:

[3] SPIFFS es un sistema de archivos diseñado para las memorias flash de los sistemas embebidos.

[4] Una flauta travesera afinada en *C* suele producir sonidos comprendidos entre la 4ª y la 6ª octava, es decir, entre 261 y 1975 Hz.

- **Tasa de muestreo**: 2000 muestras por segundo, lo que permite reconocer frecuencias de hasta 1 KHz, es decir, reconocer las notas de las 6 primeras octavas, un abanico muy amplio.
- **Número de muestras por frecuencia**: 128 muestras, lo que permite reconocer hasta 16 frecuencias por segundo [5], de nuevo, por encima de lo necesario.
- **Valor del ruido**: El micrófono escogido, en silencio, produce una señal de, aproximadamente, 500mV.

Una vez se han recogido las muestras de audio, se debe realizar el reconocimiento de las frecuencias. Para ello se aplica el algoritmo *ApproxFFT*, indicando el número de muestras tomadas y la tasa de muestreo, con lo que se obtiene la frecuencia predominante.

4.5.1.3 Configuration

El componente *configuration* **contiene** los credenciales del punto de acceso y **facilita el acceso** a ellos. Además, se trata del pilar fundamental del sistema; para que el resto de componentes funcionen, la configuración debe haber sido **previamente inicializada**. Este componente debe de realizar, obligatoriamente, las siguientes cinco operaciones, que definiremos como **operaciones fundamentales**:

Serialización de la configuración

La serialización permite convertir una instancia de la configuración en una serie de caracteres, delimitados por un carácter especial. Realizar este método facilita tanto la persistencia de la configuración en un archivo de texto, como el envío para su uso en la aplicación web.

Deserialización de la configuración

La deserialización permite convertir una serie de caracteres, cuyo delimitador es conocido, en una nueva instancia de configuración. Realizar este método facilita tanto la inicialización de la configuración desde el archivo de texto, como su actualización tras ser recibida desde la aplicación web.

Liberación de la configuración

Liberar los recursos de la configuración permite recuperar la memoria reservada para dicha instancia de configuración cuando deja de ser necesaria.

---

[5] Al utilizar una tasa de muestreo de 2000 muestras por segundo y 128 muestras por frecuencia, en un segundo se recogen $\frac{2000\,muestras}{1\,segundo} * \frac{1\,frecuencia}{128\,muestras} = 15.62\frac{frecuencia}{segundo}$, sin contar el tiempo de procesado.

Salvado de la configuración

El salvado de la configuración permite persistir en un archivo de configuración los valores establecidos por el usuario, aplicando la serialización, para su carga en posteriores inicios.

Carga de la configuración

La carga de la configuración permite crear una nueva instancia de configuración con los valores leídos desde un archivo de configuración, aplicando la deserialización.

Los métodos descritos en el código 4.1 cargan la configuración global de la misma forma, sin embargo, el segundo, además de cargar la configuración global, **carga la configuración del módulo sensorial**. Para ello, dicho método, recibe por parámetro la implementación de las cinco operaciones fundamentales, aplicadas a la configuración del módulo. Para generalizar y favorecer la escalabilidad del sistema, las operaciones fundamentales que debe implementar cada módulo sensorial están definidas según los tipos del código 4.2.

```
1.    load_configuration
2.    load_configuration_and_module
```

Código 4.1: Carga de la configuración. Métodos para realizar la carga de la configuración.

```
3.  typedef void * (*module_configuration_load)();
4.
5.  typedef int (*module_configuration_save)(void *);
6.
7.  typedef char * (*module_configuration_marshal)(void *);
8.
9.  typedef void * (*module_configuration_unmarshal)(char *);
10.
11. typedef int (*module_configuration_free)(void*);
```

Código 4.2: Definición de los tipos de las operaciones fundamentales.

Descripción de los tipos

- ***module_configuration_load***: Este tipo de función no recibe parámetros y devuelve una instancia de la configuración:
- ***module_configuration_save***: Este tipo de función recibe la instancia de configuración que se desea guardar. Devuelve *1* si el guardado se ha producido con éxito, en caso contrario, devuelve *0*.
- ***module_configuration_marshal***: Este tipo de función recibe la instancia de configuración que se desea serializar. Devuelve el texto, con la configuración serializada, o *NULL*, en caso de error.

- ***module_configuration_unmarshal***: Este tipo de función recibe un texto con una configuración serializada. Devuelve una instancia de configuración con los ajustes definidos en el texto, o NULL, en caso de error.
- ***module_configuration_free***: Este tipo de función recibe la instancia de configuración a liberar. Devuelve *1* si la liberación se ha realizado con éxito, en caso contrario, devuelve *0*.

La configuración global sólo será capaz de cargar módulos sensoriales que definan sus operaciones fundamentales según lo especificado. Al igual que los credenciales de acceso, el módulo sensorial se trata de otro parámetro más, y puede ser accedido desde la configuración global. De esta forma, **no será necesario modificar el código del sistema** para desarrollar nuevos módulos.

Para que un archivo de configuración sea cargado correctamente, cada par clave-valor debe estar delimitado por el carácter ":". Para diferenciar cada par, estos deben estar separados por un salto de línea. Por último, para asegurar el inicio del sistema, éste establece unos valores por defecto cuando no se disponga de un archivo de configuración, o éste sea erróneo.

4.5.1.4 wireless

El componente *wireless* se encarga de las comunicaciones inalámbricas, entrantes y salientes. Este componente se comporta de diferente forma, dependiendo del tipo de dispositivo:

- ***Emisor***: Se utilizan los credenciales de la configuración global para crear un punto de acceso, un *DNS* y un servidor web. Arranca en modo *WIFI_AP* [6].
- ***Cliente***: Se utilizan los credenciales de la configuración global para crear un punto de acceso, un *DNS*, un servidor web y un cliente que permite conectarse al dispositivo emisor. Arranca en modo *WIFI_AP_STA* [7]. Además los dispositivos cliente almacenan los credenciales del último dispositivo emisor al que se conectaron, lo que permite su reconexión automática tras su reinicio.

[6] *WIFI_AP* es una constante de la librería *ESP8266WiFi* e indica cómo se utilizará el módulo *WiFi*. En concreto, este valor define la creación de una red *WiFi*, a la que es posible conectarse.

[7] *WIFI_AP_STA* es una constante de la librería *ESP8266WiFi* e indica cómo se utilizará el módulo *WiFi*. En concreto, este valor incluye la funcionalidad *WIFI_AP* y, además, habilita la conexión inalámbrica a otros puntos de acceso.

Ambos dispositivos inicializan el punto de acceso, el *DNS* y el servidor web de la siguiente forma:

- El punto de acceso creado utiliza los credenciales almacenados en la configuración para crear una nueva red inalámbrica que asigna el rango de direcciones *192.168.1.0/24* a los dispositivos conectados. Por defecto, la dirección *IP* asignada al controlador es la *192.168.1.1*.
- El servidor web se inicia en el puerto 80 (el usado por *HTTP*). Este se utiliza para mantener en ejecución la aplicación web que gestiona la configuración del sistema. Para ello, durante su inicialización, se asocian los ficheros y manejadores o, *handlers*, a las rutas requeridas por la aplicación web. Estos manejadores serán detallados en la sección dedicada a la aplicación web.
- El *DNS* es configurado para redirigir todas las solicitudes realizadas al dominio *synestesia.com* a la dirección *IP 192.168.1.1*, la asignada al controlador. Éste se utiliza para acceder a la aplicación web, con el fin de mejorar la accesibilidad del sistema y la experiencia del usuario final.

4.5.1.5 postUp

El componente *postUp* se encarga del servicio inalámbrico de **retransmisión** y **recepción** de frecuencias. Para ello, hace uso de la clase *WiFiUDP* de *Arduino*, que permite enviar y recibir un conjunto de datos, encapsulados en un paquete *UDP*.

La retransmisión de la frecuencia se realiza siguiendo el modelo *publish-subscribe* [22], en el que, los dispositivos cliente, reciben la actualización de la frecuencia sin realizar alguna petición al dispositivo emisor.

Retransmisión de frecuencias

La retransmisión de frecuencias es ejecutada por el dispositivo emisor, tras el reconocimiento de cada frecuencia. Con el fin de acelerar el proceso y optimizar los recursos del dispositivo, cada frecuencia leída se encapsula en un paquete *UDP* y se envía a la dirección *broadcast* [8] de la subred creada por el componente *wireless*, permitiendo que los dispositivos conectados reciban el paquete simultáneamente. Esto supone un **único envío** por frecuencia, en lugar de uno por cliente conectado.

[8] La dirección *broadcast* se trata de la dirección de difusión de una red, en la que todos los miembros conectados reciben los mensajes enviados a ésta. La dirección utilizada por el dispositivo emisor es la *192.168.1.255*:*2512*

Recepción de frecuencias

La recepción de frecuencias es ejecutada por el dispositivo cliente sobre las comunicaciones realizadas por el dispositivo emisor, en el puerto al que son enviadas. Para evitar bloqueos durante la recepción de los mensajes del emisor, se ha establecido un tiempo de espera máximo igual a 10 ms.

4.5.1.6 LittleHashMap

El componente *LittleHashMap* se trata de una estructura de datos auxiliar en la que almacenar pares clave-valor. Este componente ha sido desarrollado para facilitar el desarrollo de los módulos sensoriales, ya que, en la gran mayoría de casos, será necesario asignar una nota con un valor. Esta estructura está diseñada para realizar la inserción y extracción de elementos mediante *hashing* y resolución de colisiones. Dada la limitación de recursos de los controladores, la estructura ha sido diseñada para almacenar las referencias de los objetos, en lugar de realizar copias de estos.

Cada par clave-valor está definido por la estructura *KeyValue*, diseñada para almacenar la clave, el valor y el *hash* de la clave, con el propósito de resolver colisiones. En concreto, el funcionamiento principal del mapa se puede resumir en tres operaciones:

Inserción

Para insertar un nuevo valor en el mapa, se genera el *hash* de la clave y, mediante la operación módulo, se obtiene la posición de inserción en la lista claves-valor. A continuación, se genera una nueva instancia de *KeyValue*, con la clave, valor y *hash*, y se inserta en la posición obtenida.

Obtención

Para obtener un valor del mapa, se genera el *hash* de la clave y, mediante la operación módulo, se obtiene la posición del valor en la lista claves-valor. Si la posición está ocupada, se devuelve el valor en dicha posición, en caso contrario, se devuelve *NULL*.

Resolución de colisiones

Durante la inserción de datos puede ocurrir que la posición de inserción obtenida esté ocupada. En este momento, se está produciendo uno de los siguientes casos:

- La clave que se está insertando ya ha sido insertada previamente y se trata de una **actualización** del valor.
- La clave que se está insertando es una nueva clave y su posición obtenida coincide con la de otra clave previamente insertada; se trata de una **colisión**.

Mediante la estructura *KeyValue*, es posible identificar cuál de las dos situaciones se está produciendo; si el *hash* del par clave-valor situado en dicha posición **es el mismo** que el *hash* de la

clave que se está insertando, se trata de una actualización: ambos objetos comparten el mismo *hash*. De lo contrario, se trata de una colisión y, es necesario obtener una nueva posición.

Al igual que en la inserción, durante la obtención de los valores puede ocurrir que la clave facilitada obtenga la posición de un par clave-valor diferente. Para evitar estas situaciones, antes de devolver el valor de una clave, se han de comprobar los *hashes.*

El algoritmo que resuelve las colisiones está implementado mediante **recursión**, pero, para evitar una recursión infinita que pueda ocasionar desbordamientos en la pila de ejecución, la recursión se limita al **tamaño máximo del mapa**, definido por el usuario durante la iniciación de la estructura.

Dado el carácter general de la estructura, es necesario indicar algunas funciones, bajo los tipos definidos en el código 4.3.

```
1.  typedef long (*hash_function)(void *);
2.
3.  typedef void (*free_map_key)(void *);
4.
5.  typedef void (*free_map_value)(void *);
```

Código 4.3: Definición de las funciones del mapa. Estas funciones hacen que el componente sea genérico.

hash_function

Esta función se utiliza para obtener el número identificativo y único (*hash*) de cada clave contenida en el mapa.

free_map_key

Esta función se utiliza para liberar las claves durante la liberación del mapa. El mapa, al contener los objetos iniciales y no copias de estos, libera el objeto original.

free_map_value

Esta función se utiliza para liberar los valores durante la liberación del mapa. El mapa, al contener los objetos iniciales y no copias de estos, libera el objeto original.

4.5.1.7 API Synestesia

La *API Synestesia* presenta la funcionalidad necesaria para integrar todos los componentes y ejecutar el sistema. Esta será detallada con profundidad en la **Sección 5**.

4.5.2 Componentes del módulo sensorial

4.5.2.1 rgb

El componente *rgb* se trata de la representación de un color *RGB*. Este objeto está compuesto por tres enteros, cuyos valores están comprendidos entre 0 y 255, e indican la cantidad de rojo, verde y azul que forman el color.

4.5.2.2 rgb_light

El componente *rgb_light* se trata de la representación de un módulo de luces *LED RGB*. Este objeto regula la tensión de los pines, definidos por configuración, a los que se encuentran conectadas las bases de los transistores y, por ende, a un color.

La regulación de la tensión aplicada a cada pin se realiza mediante *Pulse-Width-Modulation* (*PWM*), una técnica empleada para conseguir señales analógicas desde una señal digital, alternando su ciclo de trabajo. Los valores que se deben indicar para obtener una señal analógica se encuentran en el rango [0 – 255], el mismo que define la cantidad de rojo, verde y azul contenida en un objeto *RGB*. Por tanto, para crear un color en el módulo de luces *LED RGB*, se generan tres señales analógicas, una por color, cuyo valor entero corresponde a la cantidad del color que representa la conexión.

4.5.2.3 RGBLightConfiguration

El componente *RGBLightConfiguration* **contiene un conjunto de luces**, en el que, cada nota musical corresponde a un color. Dicha asignación es **realizada por el usuario en tiempo de ejecución**, desde la aplicación web, donde, además, es posible definir nuevas luces, cada una, con un mapa de colores independiente, representado en la ilustración 4.3.

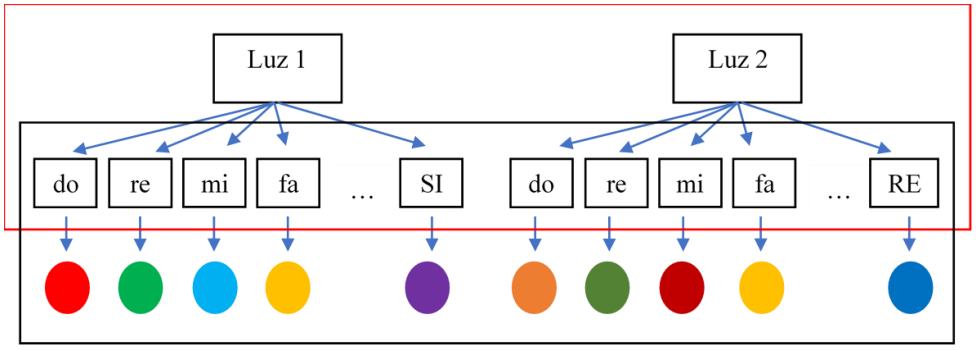

Ilustración 4.3: Mapa de colores por nota y por luz

El recuadro rojo señala el mapa utilizado para la asociación de luces y colores **por nota**. Es decir, dada una **luz**, es posible obtener el **mapa de notas** al que referirse para extraer el color que debe ser mostrado. Dicho mapa de notas, marcado en negro, es utilizado para la **asociación de notas y colores**, en el que, mediante una nota musical, es posible obtener el color asignado por el usuario. Este mapa es independiente para cada luz, lo que permite realizar asociaciones de colores por nota diferentes entre luces. Para llevar a cabo esta implementación se ha hecho uso de la estructura *LittleHashMap*, anteriormente explicada, con las funciones *hashing*:

- **Mapa de luces**: Una luz está identificada por sus conexiones, por lo que, su función hash está definida por la concatenación de los números de los pines en los que está conectada.
- **Mapa de notas**: Las notas están definidas por su posición, desde el 1, hasta la última posición en la última octava reconocida.

Para realizar la conversión de frecuencia a nota musical, este componente ofrece un método que traduce una frecuencia leída a una nota musical (ver anexo D), limitada por el número de octavas definidas, mediante la operación módulo (*%*).

Para que la configuración del módulo pueda ser utilizada en el sistema, debe ser iniciada desde la configuración principal, indicando, por argumento, la implementación realizada de las **operaciones fundamentales**, definidas anteriormente. Esta limitación se ha implantado para asegurar la existencia y correcta definición de dichas operaciones, necesarias para la comunicación entre sistema y el módulo sensorial (serialización, guardado desde la web, etc.).

Dada la complejidad de *deserializar* la estructura del módulo, este método se ha definido según el diagrama de flujo de la ilustración 4.4.

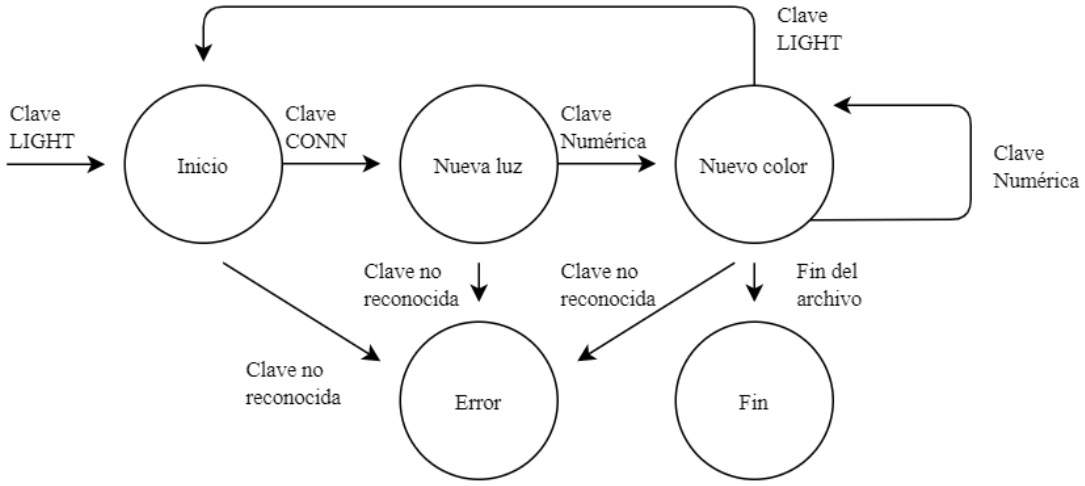

Ilustración 4.4: Diagrama de flujo del procesado de un archivo de configuración del módulo sensorial.

4.5.3 Aplicación web

La aplicación web permite al usuario configurar el sistema y persistir los cambios en el tiempo, sin que sea necesaria la modificación del código. Esta aplicación se ejecuta en el puerto 80 del servidor de ambos dispositivos, emisor y cliente, lo que hace posible el acceso a la configuración desde cualquier dispositivo con un navegador web.

La aplicación, retratada en las imágenes del anexo C, está basada en el concepto *SPA* y se encuentra programada en *javascript*, con la ayuda del *framework Vue*, que facilita la creación de *HTML* dinámico. Además, para agilizar el desarrollo y mejorar la accesibilidad, se ha hecho uso de los estilos *CSS* proporcionados por *Bootstrap*. Estas fuentes han sido cargadas, en su versión minificada al sistema de archivos del controlador, ya que no es posible que el usuario los obtenga de los *CDNs*, al carecer de una conexión a internet cuando se encuentra conectado al punto de acceso del controlador.

Por otro lado, ha sido necesario desarrollar los componentes web:

- **Scripts.js:** Este componente contiene funciones de utilidad comunes a todos los componentes, como realizar peticiones asíncronas.
- **Global.js:** Este componente se trata de un componente *Vue*, de nombre *Global*. Representa la configuración global y gestiona la obtención y **deserialización** del objeto *configuration*.
- **Module.js:** Este componente se trata de un componente *Vue*, de nombre *Module*. Representa la configuración del módulo y gestiona la obtención **deserialización** del objeto *RGBLightConfiguration*.
- **Receiver.js:** Este componente se trata de un componente *Vue*, de nombre *Receiver*, disponible únicamente en los dispositivos cliente. Habilita un formulario donde se introducen los credenciales del punto de acceso.
- **Configuration.html**: Este componente se trata de un contenedor *HTML*, además, contiene la aplicación *Vue* principal, que instancia los componentes *Global* y *Module*.
- **Configuration-receiver.html:** Este componente se trata del mismo que el componente *Configuration.html*, con la excepción de que este componente, además, instancia el componente *Receiver* y, por tanto, está solo disponible en los dispositivos cliente.

Utilizando esta distribución de ficheros, los desarrolladores no tendrán que modificar el código del componente *wireless* para incluir nuevos módulos sensoriales, sólo sustituir el contenido del archivo *module.js*.

Para terminar, la comunicación entre los componentes web y el sistema, se realiza mediante controladores o *endpoints*, definidos en el componente *wireless*, para escuchar las peticiones realizadas a las rutas:

/configuration

- ***GET***: Permite la carga del *HTML* principal; *configuration.html*, para los dispositivos emisor y, *configuration-receiver.html*, para los dispositivos cliente.
- ***POST***: Persiste la configuración global y del módulo, si está presente.

/global

- ***GET***: Permite la carga del componente web de la configuración global.

/module

- ***GET***: Permite la carga del componente web de la configuración del módulo, si está presente. De lo contrario, devuelve el error *404 – Not Found*.

/global-data

- ***GET***: Solicita la serialización de la configuración global actual, para enviarla en formato de texto.

/module-data

- ***GET***: Solicita la serialización de la configuración del módulo actual, para enviarla en formato de texto.

/reset

- ***GET***: Resetea el controlador. Es necesario para aplicar la configuración modificada.

/connect (Sólo dispositivo cliente)

- ***POST***: Realiza una conexión a un dispositivo emisor, utilizando los credenciales indicados en el formulario del componente *Receiver*.

4.6 *Estructura del proyecto*

Para organizar correctamente la estructura del proyecto, este ha sido distribuido en varios directorios, contenidos dentro del directorio *synestesia*. El contenido de dichos directorios ha sido distribuido según la funcionalidad que aporte al sistema:

4.6.1 Componentes software

Son los componentes desarrollados para añadir funcionalidad al sistema. Al igual que durante su desarrollo, es posible distinguirlos en dos tipos; principales (*main*) y del módulo (*module*). Estos componentes se encuentran dentro del directorio ***synestesia/libs***, contenidos en **paquetes**, donde se ubican dos ficheros, uno con el código fuente y otro con las cabeceras de los métodos públicos.

4.6.2 Programas de prueba

Por cada componente desarrollado se ha realizado un programa de prueba. Estos programas de pruebas se encuentran definidos en la carpeta ***synestesia/test***, bajo el nombre y código de la tarea, anotada en el tablero *Kanban*, que ha implicado el desarrollo del componente a probar.

4.6.3 Programas principales

Los programas principales son los programas a ejecutar en los controladores *NodeMCU* y *Arduino Nano*; estos contienen el sistema en su totalidad, listo para su ejecución. Dichos programas se encuentran en la carpeta ***synestesia/main***.

4.6.4 Ficheros auxiliares

Los ficheros auxiliares son los ficheros que, necesariamente, han de ser cargados al controlador, para la ejecución de la aplicación web. Estos ficheros se encuentran en la carpeta *data*, contenida en los programas principales del dispositivo emisor y cliente.

5 Implementación

La integración de todos los componentes desarrollados se realiza mediante la *API Synestesia*. Esta interfaz se ha desarrollado con la intención de facilitar la ejecución del sistema y el desarrollo de módulos sensoriales.

5.1 *Iniciación del sistema*

Para añadir consistencia al sistema, se ha implementado el patrón *Singleton*, con el que sólo es posible realizar una iniciación del objeto principal y, por tanto, tener una instancia de *Synestesia* al mismo tiempo.

Como el sistema está compuesto por dos tipos de controladores, la iniciación puede realizarse de dos formas diferentes.

5.1.1 Controladores NodeMCU

Los controladores *NodeMCU* pueden incorporar un módulo sensorial, por tanto, la definición del método debe incluir una forma de indicar la implementación de las cinco operaciones fundamentales. Para ello, se hace uso de la estructura *ModuleFunctions*, una estructura diseñada para envolver las operaciones fundamentales en un único objeto. Por otro lado, como ambos controladores utilizan el mismo método, definido en el código 5.1, para diferenciar qué tipo de controlador está iniciando el sistema (emisor o cliente), se debe indicar el *DeviceType*, una enumeración que permite realizar unas u otras funciones, dependiendo del dispositivo:

- *MASTER*: Define el controlador *NodeMCU* del **dispositivo emisor**.
- *SLAVE*: Define el controlador *Arduino Nano* del **dispositivo emisor**.
- *RECEIVER*: Define el controlador *NodeMCU* del **dispositivo cliente**.
- *NONE*: Define un dispositivo desconocido, usado para el **control de errores**.

```
1.  Synestesia * initialize(ModuleFunctions * mFn, DeviceType type);
```

Código 5.1: Método de inicio de Synestesia para controladores *NodeMCU*.

Dado que el módulo sensorial es opcional, no es necesario definir la estructura *ModuleFunction*, y puede indicarse como *NULL*. El método valida la estructura y cada uno de sus campos, y, en caso de encontrar un fallo, inicia el sistema sin definir un módulo sensorial. Por otro lado, este método no acepta otro *DeviceType* que no sea *MASTER* o *RECEIVER*.

5.1.1.1 MASTER

El dispositivo *master* realiza la carga de las configuraciones y posteriormente, inicia las dependencias inalámbricas, es decir, el componente *wireless* y el servicio *postUp,* además de la comunicación alámbrica con el controlador auxiliar, al que solicitará la frecuencia analizada por éste, mediante la comunicación I^2C.

5.1.1.2 RECEIVER

El dispositivo *receiver* realiza la carga de las configuraciones y, posteriormente, inicia las dependencias inalámbricas, es decir, el componente *wireless* y el servicio *postUp*.

5.1.2 Controlador Arduino Nano

El controlador *Arduino Nano* se utiliza para analizar las frecuencias sonoras y no requiere de ninguna configuración por parte del usuario, por lo que el método utilizado, el descrito en el código 5.2, no recibe ningún parámetro e infiere el tipo de dispositivo (*SLAVE*) automáticamente

```
2.  Synestesia * initialize_slave();
```

Código 5.2: Método de inicio de Synestesia para el controlador *Arduino Nano.*

El dispositivo *SLAVE* inicializa la comunicación alámbrica con el controlador principal, con el que compartirá la frecuencia analizada cuando éste la solicite.

5.2 *Ejecución del sistema*

La ejecución del sistema es idéntica para los tres controladores y es realizada mediante la las llamadas reiteradas a un mismo método, definido en el código 5.3, lo que permite que el sistema pueda ser ejecutado en paralelo con otras tareas del usuario.

```
3.  float run_core(Synestesia * synestesia);
```

Código 5.3: Método de ejecución de Synestesia.

Este método de ejecución recibe la instancia de la interfaz *Synestesia* y devuelve la última frecuencia analizada. Internamente, se realiza la diferenciación del dispositivo mediante el tipo asignado durante la inicialización y, dependiendo de éste, el flujo de ejecución es diferente. Sin embargo, en ninguno de los flujos se realiza la ejecución de las funciones del módulo sensorial, lo que permite la ejecución de acciones más específicas, en lugar de estar acotadas por una definición genérica.

5.2.1 DeviceType – MASTER

Durante la ejecución del dispositivo *MASTER*, se realiza la gestión del servidor web, la obtención de las frecuencias y su retransmisión mediante el servicio *postUp*. Mientras que la gestión de las peticiones web es continua en la ejecución, la obtención de frecuencias y su retransmisión se realiza mediante la técnica conocida como *Protothreading*, que permite su ejecución intermitente, reduciendo la carga de trabajo contenida en cada ejecución, sin que sea notorio para el usuario.

5.2.2 DeviceType – SLAVE

Durante la ejecución del dispositivo *SLAVE* se realiza el análisis de las frecuencias leídas por la interfaz de audio, mediante el componente *audio*. Al **recibir una solicitud** de obtención de frecuencia, este controlador envía la última frecuencia analizada al controlador principal.

5.2.3 DeviceType – RECEIVER

Durante la ejecución del dispositivo *RECEIVER*, se realiza la gestión del servidor web y la recepción de las frecuencias mediante el servicio *postUp*.

5.3 *Ejecución del Módulo sensorial*

La ejecución del módulo sensorial se realiza tras haber obtenido la frecuencia analizada durante la ejecución del sistema principal. Esta frecuencia es utilizada para obtener la nota a la que corresponde y, posteriormente, mostrar los colores de dicha nota en cada una de las luces del sistema.

6 Pruebas y resultados

6.1 *Pruebas unitarias*

Las pruebas unitarias se encuentran en el directorio *synestesia/test* y comprueban el funcionamiento individual de los componentes desarrollados. Para ello, se han desarrollado los programas de pruebas:

- ***SYN-1-rgb-lights***: En esta prueba se verifica el funcionamiento de los componentes *rgb* y *rgb_light*, iluminando el módulo de luces *LED*, en diferentes colores.
- ***SYN-6-global-config***: En esta prueba se verifica el funcionamiento del componente *configuration*, realizando la carga desde un fichero y su posterior persistencia.
- ***SYN-15-file-manager***: En esta prueba se verifica el funcionamiento del componente *file_manager*, mediante la lectura y escritura de un fichero.
- ***SYN-16-module-configuration***: En esta prueba se verifica el funcionamiento del componente *RGBLightConfiguration*, realizando la carga desde un fichero, su persistencia y la obtención varios colores dadas una luz y algunas notas.
- ***SYN-20-web-access***: En esta prueba se verifica el funcionamiento de la aplicación web, para ello, el usuario debe lanzar el programa, conectarse al controlador y acceder a ella, en la dirección ***synestesia.com/configuration***.
- ***SYN-21-access-point***: En esta prueba se verifica el funcionamiento del componente *wireless*, mediante la creación de una red *WiFi* con los valores almacenados en la configuración.
- ***SYN-23-signal-management***: En esta prueba se verifica el funcionamiento del componente *audio*, mediante el análisis de sonidos y la representación de la frecuencia predominante en la consola.
- ***SYN-24-handle-clients-and-signals***: En esta prueba se verifica el funcionamiento de la comunicación I^2C entre el controlador *NodeMCU* y *Arduino Nano*. Esta prueba consiste en dos programas, cada uno debe ser ejecutado en cada controlador
- ***SYN-26-udp-service***: En esta prueba se verifica el funcionamiento del componente *postUp*, mediante la retransmisión de las frecuencias a la dirección de broadcast.
- ***SYN-27-synestesia-manager***: En esta prueba se verifica la integración de todos los componentes y su correcto funcionamiento. Esta prueba consiste en tres programas, dos para los controladores *NodeMCU* y uno para el controlador *Arduino Nano*.

6.2 *Reconocimiento de frecuencias*

En esta prueba se comprueba el análisis de las frecuencias y la obtención de las notas asociadas. Para llevar a cabo esta prueba, se han analizado con un afinador las frecuencias de las notas de las dos primeras octavas de una flauta travesera y, posteriormente, se han recogido las frecuencias y notas musicales reconocidas por el sistema, resultando las anotadas en la ***tabla 6.1***. En esta tabla se han marcado los aciertos en verde y, los errores, en rojo. En la gran mayoría de casos, el sistema reconoce correctamente la nota producida por el instrumento, a pesar de que las frecuencias reconocidas disten unos pocos hercios de las analizadas por el frecuencímetro digital.

Nota interpretada	Frecuencia analizada (Hz)	Frecuencia leída por el sistema (Hz)	Nota musical obtenida por el sistema
Do / C	261	265	do
Re / D	291	296	re
Mi / E	328	331	mi
Fa / F	350	356	fa
Sol / G	396	402	sol
La / A	441	447	la
Si / B	501	512	do
Do / C	533	545	do
Re / D	599	609	re
Mi / E	673	687	mi
Fa / F	711	718	fa
Sol / G	797	812	sol
La / A	904	909	la
Si / B	980	970	si

Tabla 6.1: Comparativa de las notas musicales y frecuencias.

6.3 *Retransmisión de las frecuencias*

Para verificar que la retransmisión de las frecuencias mediante la dirección de difusión es correcta, se ha realizado un análisis del tráfico de paquetes en la red del emisor, con la herramienta *Wireshark*. Tal como se esperaba, estos son emitidos desde el dispositivo emisor, *192.168.1.1:2512*, a la dirección de broadcast, *192.168.1.255:2512*, estando disponibles para todos los dispositivos conectados. En el anexo E, los paquetes difundidos se muestran en detalle.

7 Conclusiones y trabajo futuro

7.1 *Conclusiones*

El desarrollo de este sistema ha supuesto un gran reto; no sólo ha sido necesario el desarrollo de un proyecto destinado a ser completamente configurable y escalable, incluyendo hardware y software, sino también, ha sido necesario afianzar los conocimientos relacionados con las frecuencias, su tratamiento y las transformaciones que pueden sufrir.

Otra de las dificultades, ha sido realizar el sistema con dispositivos económicos, de bajo consumo y fáciles de obtener, lo que ha supuesto complicaciones durante el desarrollo de los componentes software, específicamente desarrollados para proporcionar un buen rendimiento durante la ejecución, lo que requiere de diseños software elaborados y "respetuosos" con los recursos.

Sin embargo, tal como se propuso inicialmente, ha sido posible realizar los dos dispositivos prototipo, emisor y cliente, con la funcionalidad necesaria para su uso indiscreto. Estos prototipos se encuentran muy cerca de un producto final que podría convertirse en un objeto esencial en las interpretaciones musicales, para ofrecer, a **todos** los asistentes, nuevas formas de sentir la música.

Por último, es importante destacar el potencial de este sistema, ya que puede ser ampliamente explotado por la comunidad *maker*, quienes podrán extenderlo y completarlo libremente. Para ello, todo el código desarrollado se encuentra disponible en un repositorio de *GitHub*:

https://github.com/Juanjod54/Synestesia

7.2 *Trabajo futuro*

El trabajo futuro consistirá, principalmente, en el desarrollo de nuevos módulos sensoriales, así como la expansión del sistema principal para recoger todas las necesidades que puedan surgir con estos.

Por otro lado, sería necesario solucionar los defectos conocidos por el momento, como mejorar la precisión del algoritmo que realiza la conversión de frecuencias a notas musicales, además de añadir el reconocimiento de notas intermedias (bemoles).

Otra funcionalidad que ayudará a la expansión del sistema será permitir la exportación e importación de archivos de configuración desde la web, permitiendo aplicar una configuración a varios dispositivos rápida y cómodamente. Además, será necesario restringir el acceso a la configuración de la web, con una contraseña.

En términos generales, sería deseable convertir los prototipos en un producto final, alimentados con una batería externa, para añadir mayor libertad de movimiento, al igual que encapsular los controladores en una carcasa, para facilitar su manipulación.

Por último, para lograr el desarrollo de módulos sensoriales por parte de terceros, será imprescindible añadir un manual detallado de todo el sistema.

8 Referencias

[1] 3Blue1Brown, «Youtube,» [En línea]. Available: https://www.youtube.com/watch?v=spUNpyF58BY.

[2] Wikipedia, «Wikipedia,» 22 01 2021. [En línea]. Available: https://es.wikipedia.org/wiki/Transformada_r%C3%A1pida_de_Fourier.

[3] A. Patel, «Instructables,» 12 2020. [En línea]. Available: https://www.instructables.com/ApproxFFT-Fastest-FFT-Function-for-Arduino/.

[4] Amilcare, «elettroamici,» 18 06 2020. [En línea]. Available: https://www.elettroamici.org/en/teorema-di-nyquist-shannon/.

[5] Arduino, «Arduino,» [En línea]. Available: https://www.arduino.cc/en/Guide/Introduction.

[6] Microchip, «Microchip,» [En línea]. Available: https://www.microchip.com/wwwproducts/en/ATmega328P.

[7] Espressif, «Espressif,» [En línea]. Available: https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf.

[8] S. Marley, «GitHub,» 2020. [En línea]. Available: https://github.com/s-marley/ESP32_FFT_VU.

[9] cnlohr, «Blogspot,» 18 11 2010. [En línea]. Available: http://cnlohr.blogspot.com/2010/11/colorchord-sound-lighting.html.

[10] Wikipedia, «Wikipedia,» 24 05 2021. [En línea]. Available: https://en.wikipedia.org/wiki/Discrete_Fourier_transform.

[11] cnlohr, «GitHub,» [En línea]. Available: https://github.com/cnlohr/colorchord.

[12] E. C. Forum, «GitHub,» ESP8266 Community Forum, 2015. [En línea]. Available: https://github.com/esp8266/Arduino.

[13] I. Grokhotkov, «arduino-esp8266,» 2017. [En línea]. Available: arduino-esp8266.

[14] E. C. Forum, «GitHub,» [En línea]. Available: https://github.com/esp8266/Arduino/blob/master/libraries/ESP8266WiFi/src/WiFiUdp.h.

[15] E. C. Forum, «GitHub,» [En línea]. Available: https://github.com/esp8266/Arduino/tree/master/libraries/ESP8266WebServer.

[16] E. C. Forum, «GitHub,» [En línea]. Available: https://github.com/esp8266/Arduino/blob/master/libraries/DNSServer/examples/DNSServer/DNSServer.ino.

[17] E. C. Forum, «GitHub,» [En línea]. Available: https://github.com/esp8266/Arduino/blob/master/doc/filesystem.rst.

[18] I. Seidel, «GitHub,» [En línea]. Available: https://github.com/ivanseidel/ArduinoThread.

[19] Vue.js, «Vue.js,» [En línea]. Available: https://vuejs.org/.

[20] B. Team, «Bootstrap,» [En línea]. Available: https://getbootstrap.com/.

[21] S. Campbell, 2016. [En línea]. Available: https://www.circuitbasics.com/basics-of-the-i2c-communication-protocol/.

[22] Wikipedia, «Wikipedia,» 15 05 2021. [En línea]. Available: https://en.wikipedia.org/wiki/Publish%E2%80%93subscribe_pattern.

[23] CNLohr, «Blogspot,» 18 11 2010. [En línea]. Available: http://cnlohr.blogspot.com/2010/11/colorchord-sound-lighting.html.

[24] N. B. Root, R. Rouw, M. Asano, C.-Y. Kim, H. Melero, K. Yokosawa y V. S. Ramachandran, «Why is the synesthete's “A” red? Using a five-language dataset to

disentangle the effects of shape, sound, semantics, and ordinality on inducer–concurrent relationships in grapheme-color synesthesia,» *ScienceDirect,* vol. 99, pp. 375-389, 2018.

[25] Banco~commonswiki, «Wikimedia Commons,» 27 12 2005. [En línea]. Available: https://upload.wikimedia.org/wikipedia/commons/0/03/Mic-condenser.PNG.

[26] Fritzing, «Fritzing,» [En línea]. Available: https://fritzing.org/.

[27] Arduino, «Arduino,» 05 02 2018. [En línea]. Available: https://www.arduino.cc/en/Tutorial/Foundations/PWM.

Glosario

API	Application Programming Interface
CDN	Content Delivery Network
SPA	Single Page Application
RGB	Red, Green, Blue
LED	Light-Emitting Diode
UDP	User Datagram Protocol
HTTP	Hypertext Transfer Protocol
MIPS	Million Instructions Per Second
GPIO	General Purpose Input/Output
CSS	Cascading Style Sheets
I^2C	Inter-Integrated Circuit
IP	Internet Protocol
WiFi	Wireless Fidelity
FFT	Fast Fourier Transform
TCP	Transmission Control Protocol
CPU	Central Processing Unit

Anexos

A Manual de instalación

Pasos previos

Para realizar la instalación del sistema es necesario haber configurado el entorno, por lo que, previamente, se han de cumplir los siguientes requisitos:

- Se debe haber instalado el entorno de desarrollo integrado de *Arduino*.
- Se debe haber instalado el proyecto *Arduino ESP8266 Core*.
- Se debe haber instalado la herramienta *Arduino ESP8266 filesystem uploader*
- Se debe haber instalado la librería *ArduinoThread*.
- Se debe haber realizado el montaje de los dispositivos emisor y cliente, según lo indicado en la **ilustración 4.2**.

Además, es necesario disponer del código fuente, disponible en la última *release,* del enlace:

https://github.com/Juanjod54/Synestesia/releases

Instalación

Dentro del fichero descargado, se encuentra el contenido del proyecto *Synestesia*. Para realizar la instalación del sistema, es necesario añadir los componentes del proyecto a las librerías de *Arduino* y, además, cargar los programas principales en el dispositivo correspondiente.

Añadir los componentes

Para añadir los componentes desarrollados basta con extraer las carpetas *libs/main* y *libs/module* en el directorio *libraries* de Arduino.

Carga de los programas principales

La carga de los programas principales consiste en la subida del *sketch* al controlador, para que éste pueda ejecutarlo correctamente.

- *MASTER.ino*: Este programa ha de ser cargado en el controlador *NodeMCU* del dispositivo emisor.
- *SLAVE.ino*: Este programa ha de ser cargado en el controlador *Arduino Nano* del dispositivo emisor.

- *RECEIVER*.ino: Este programa há de ser cargado en el controlador *NodeMCU* del dispositivo receptor.

Además, también será necesario añadir los ficheros auxiliares a la memoria flash del controlador. Para ello, sólo es necesario hacer clic sobre la opción de la herramienta, ya que cada programa principal incluye el directorio *data*, con dichos ficheros.

B Tareas por sprint

En este anexo se detalla la lista de tareas realizadas para el desarrollo del sistema, por *sprints*. En concreto, la tabla B.1 contiene el código de la tarea y las fechas en las que se ha realizado y, la ilustración B.1, contiene el desarrollo de éstas a lo largo de cada *sprint.*

Sprint	Tarea	Fecha inicio	Fecha fin
1	**SYN-1**: Gestión Luces RGB	23/03/2021	28/03/2021
1	**SYN-2**: Selección de color basado en entrada	29/03/2021	01/04/2021
2	**SYN-6**: Añadir sistema de configuración genérico	12/04/2021	14/04/2021
2	**SYN-13**: Definir formato tipo .syn genérico	11/04/2021	11/04/2021
2	**SYN-14**: Definir formato tipo .syn RGB_LIGHT	11/04/2021	11/04/2021
2	**SYN-15**: Crear sistema de gestión de archivos	11/04/2021	23/04/2021
2	**SYN-16**: Añadir sistema de configuración RGB_LIGHT	13/04/2021	25/04/2021
3	**SYN-20**: Acceder a las configuraciones desde la web	26/04/2021	03/05/2021
3	**SYN-21**: Crear servidor WEB accesible	26/04/2021	03/05/2021
3	**SYN-22**: Persistir la configuración en archivos syn	04/05/2021	14/05/2021
4	**SYN-23**: Librería de procesado de señales	15/05/2021	15/05/2021
4	**SYN-24**: Implementar configuración y procesado de señales bajo el mismo programa	15/05/2021	17/05/2021
4	**SYN-25**: Mejoras	18/05/2021	18/05/2021
4	**SYN-26**: Servicio UDP	17/05/2021	18/05/2021
5	**SYN-27**: Crear clase Synestesia para que gestione las modalidades correctamente	18/05/2021	29/05/2021
5	**SYN-28**: Crear modalidad emisor	29/05/2021	29/05/2021
5	**SYN-29**: Crear modalidad receptor	29/05/2021	29/05/2021

Tabla B.1: Lista de las tareas, junto a las fechas de realización.

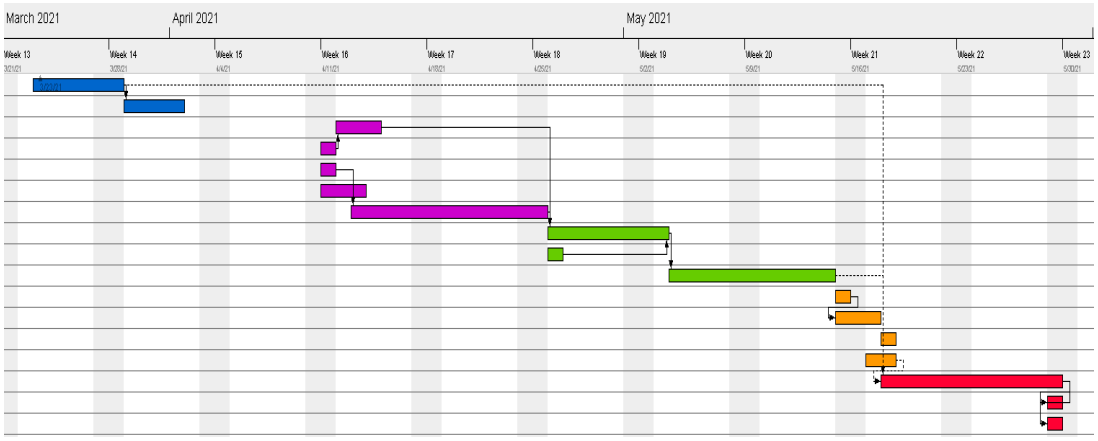

Ilustración B.1: Disposición de las tareas a lo largo de los sprint.

C Aplicación web

En este anexo se adjuntan tres capturas de pantalla, como demostración de la aplicación web desarrollada para configurar el sistema.

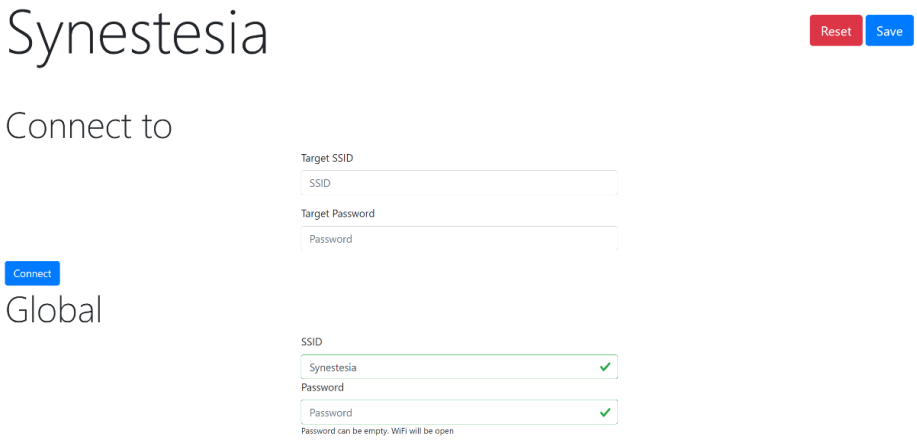

Ilustración C.1: Vista de la configuración de un dispositivo cliente.

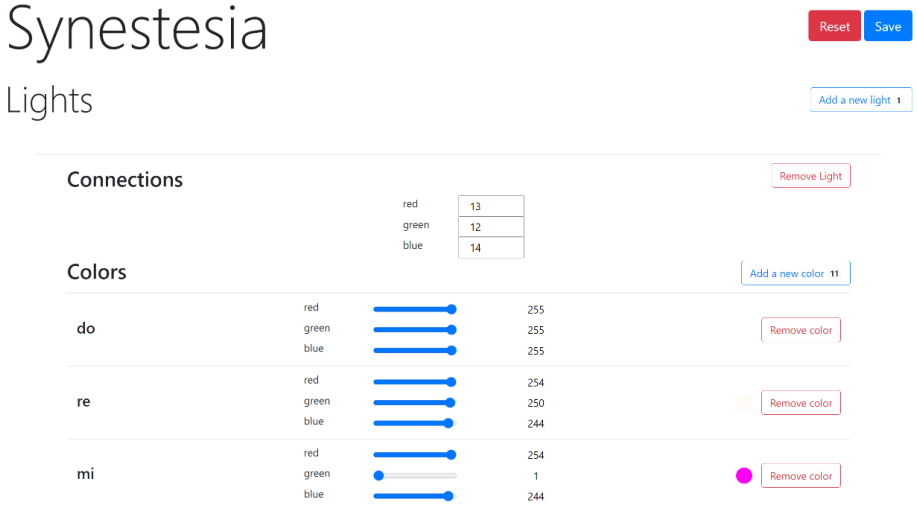

Ilustración C.2: Vista de la configuración del módulo de luces.

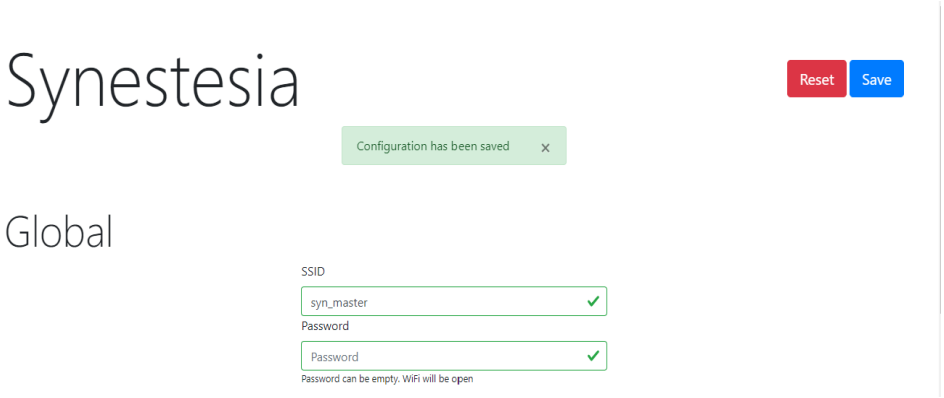

Ilustración C.3: Vista de una actualización correcta.

D Conversión de frecuencia a nota musical

Para realizar correctamente la conversión de frecuencia a nota musical, es necesario extraer algunos puntos claves de la tabla D.1:

	Octava 0	Octava 1	Octava 2	Octava 3	Octava 4	Octava 5	Octava 6	Octava 7	Octava 8
Do	16.3516	32.7032	65.4064	130.8128	261.6256	523.2512	1046.502	2093.005	4186.01
Re	18.3500	36.7	73.4	146.8	293.6	587.2	1174.4	2348.8	4697.6
Mi	20.6017	41.2034	82.4068	164.8136	329.6272	659.2544	1318.509	2637.018	5274.035
Fa	21.8268	43.6536	87.3072	174.6144	349.2288	698.4576	1396.915	2793.83	5587.661
Sol	24.4997	48.9994	97.9988	195.9976	391.9952	783.9904	1567.981	3135.962	6271.923
La	27.5000	55	110	220	440	880	1760	3520	7040
Si	30.8677	61.7354	123.4708	246.9416	493.8832	987.7664	1975.533	3951.066	7902.131

Tabla D.1: Frecuencias por nota y octava.

- La frecuencia de una nota en la Octava X es la mitad del valor de la misma nota en la Octava X + 1.
- La frecuencia de una nota en la Octava X es **dos veces** la mitad del valor de la misma nota en la Octava X + 2
- La frecuencia de una nota en la Octava X es la **Y veces** la mitad del valor de la misma nota en la Octava X + Y, (X+Y > X).

De este desarrollo se puede inferir la ecuación D.1, donde *f (n, x)* significa la frecuencia de la nota N (do, re, mi), en la octava x (0, 1)

$$f_{(n,\ x+y)} = f_{(n,\ x)} * 2^{y} \qquad \textbf{Ec D.1}$$

Por ejemplo:

$$f_{(Do,\ 0)} = 16.35\ Hz \qquad \textbf{Ec D.2}$$

$$f_{(Do,\ 5)} = 523.2512\ Hz \qquad \textbf{Ec D.3}$$

$$f_{(Do,\ 0+5)} = 2^{5} * 16.35\ Hz = 523.2\ Hz \qquad \textbf{Ec D.4}$$

De igual forma, es posible obtener la octava a la que pertenece una frecuencia, mediante la ecuación D.5.

$$\frac{f_{(n,\ x+y)}}{f_{(n,\ x)}} = 2^{y} \qquad \textbf{Ec D.5}$$

La ecuación D.5 resulta de gran interés para transportar la frecuencia a otra octava y, obtener la distancia con la frecuencia más baja de dicha octava, mediante la ecuación D.6, donde *f(lowest, x)*, indica la frecuencia más baja de la octava

$$Distance = \frac{f(n, x+y)}{2^{y}} - f(lowest, x) \qquad \textbf{Ec D.6}$$

La distancia secuencial entre dos notas consecutivas de una misma escala es, aproximadamente, 2^{x+1}. Esta distancia, permite conocer la nota musical a la que corresponde la frecuencia que, junto a los cálculos anteriores, han caracterizado una frecuencia en una nota y una octava.

E Capturas de paquetes con Wireshark.

Para realizar la captura de los paquetes emitidos por el dispositivo emisor, se ha hecho uso del programa Wireshark. En él, es posible capturar los paquetes que circulan por una red, en este caso, la del dispositivo emisor. Cada paquete capturado corresponde a una retransmisión a la dirección de broadcast, con la frecuencia leída. Como se comentó durante la sección 5, el análisis de frecuencia se realiza en una ejecución intermitente y, por tanto, no se actualiza a la misma velocidad a la que se retransmiten los paquetes.

En la tabla E.1 se registran los paquetes retransmitidos en un intervalo de 3 ms, indicando el tiempo en el que se recibieron en la interfaz de red del ordenador, el retraso entre paquetes, la IP de origen y destino, el protocolo y la frecuencia retransmitida. Tal como se puede apreciar, el retraso entre paquetes es mínimo, lo que aumenta la posibilidad de que los dispositivos clientes reciban al menos un paquete por cada frecuencia (UDP no garantiza que los datos sean recibidos).

Time	Delay (µs)	Source	Destination	Protocol	Frequency (Hz)
00:03:51,299370	0	192.168.1.1	192.168.1.255	UDP	938,73
00:03:51,302096	0,03	192.168.1.1	192.168.1.255	UDP	938,73
00:03:51,304588	0,03	192.168.1.1	192.168.1.255	UDP	938,73
00:03:51,306983	0,02	192.168.1.1	192.168.1.255	UDP	938,73
00:03:51,309847	0,03	192.168.1.1	192.168.1.255	UDP	938,73
00:03:51,310700	0,01	192.168.1.1	192.168.1.255	UDP	938,73
00:03:51,312605	0,02	192.168.1.1	192.168.1.255	UDP	938,73
00:03:51,313586	0,01	192.168.1.1	192.168.1.255	UDP	938,73
00:03:51,315650	0,02	192.168.1.1	192.168.1.255	UDP	938,73
00:03:51,318779	0,03	192.168.1.1	192.168.1.255	UDP	938,73
00:03:51,321223	0,02	192.168.1.1	192.168.1.255	UDP	938,73
00:03:51,517191	2,27	192.168.1.1	192.168.1.255	UDP	938,73
00:03:51,519381	0,02	192.168.1.1	192.168.1.255	UDP	938,73
00:03:51,522141	0,03	192.168.1.1	192.168.1.255	UDP	938,73
00:03:51,524442	0,02	192.168.1.1	192.168.1.255	UDP	938,73
00:03:51,526670	0,03	192.168.1.1	192.168.1.255	UDP	938,73
00:03:51,532073	0,06	192.168.1.1	192.168.1.255	UDP	938,73
00:03:51,533907	0,02	192.168.1.1	192.168.1.255	UDP	938,73
00:03:51,536740	0,03	192.168.1.1	192.168.1.255	UDP	562,76
00:03:51,538928	0,02	192.168.1.1	192.168.1.255	UDP	562,76
00:03:51,541898	0,03	192.168.1.1	192.168.1.255	UDP	562,76
00:03:51,544077	0,02	192.168.1.1	192.168.1.255	UDP	562,76
00:03:51,546668	0,03	192.168.1.1	192.168.1.255	UDP	562,76
00:03:51,549376	0,02	192.168.1.1	192.168.1.255	UDP	562,76
00:03:51,551460	0,02	192.168.1.1	192.168.1.255	UDP	562,76
00:03:51,554052	0,03	192.168.1.1	192.168.1.255	UDP	562,76
00:03:51,556699	0,03	192.168.1.1	192.168.1.255	UDP	562,76
00:03:51,558965	0,02	192.168.1.1	192.168.1.255	UDP	562,76
00:03:51,561559	0,03	192.168.1.1	192.168.1.255	UDP	562,76
00:03:51,564891	0,03	192.168.1.1	192.168.1.255	UDP	562,76
00:03:51,566436	0,01	192.168.1.1	192.168.1.255	UDP	562,76
00:03:51,569095	0,03	192.168.1.1	192.168.1.255	UDP	562,76
00:03:51,571390	0,02	192.168.1.1	192.168.1.255	UDP	562,76
00:03:51,575182	0,05	192.168.1.1	192.168.1.255	UDP	562,76
00:03:51,577304	0,02	192.168.1.1	192.168.1.255	UDP	562,76
00:03:51,579689	0,03	192.168.1.1	192.168.1.255	UDP	562,76

Tabla E.1: Resumen de paquetes con *Wireshark*.